Appendix A

Title: Harrier-VT

CMOS

3 Overview

F-TQFP-?

The HARRIER-VT provides an highly integrated single chip solution for Voice-over-IP (VoIP) and Fax-over-IP (FoIP) applications. This solution has been tailored to serve applications such as VoIP cable modems, residential gateways, LAN telephones and LAN adapters.

The HARRIER-VT is based on Infineon's latest 32-bit TriCore architecture. This high end 32-bit single core CISC/RISC/DSP with dual MAC and SIMD capability provides a MIPS budget, which is suitable to run both, high compression/low-delay audio codecs such as G.7xx, as well as VoIP protocol stacks such as H.323, x.GCP, or SIP on a single device.

Real-time operating system support including VxWorks, Nucleus+ and PSOS provides access to a wide range of commercial and proprietary application software products.

The architecture is based on the Flexible Peripheral Interface Bus (FPI), which is interconnecting the embedded 32-bit TriCore with a variety of dedicated communication modules. Besides standard asynchronous and synchronous serial interfaces (USART, IrDA, SPI) the HARRIER-VT provides interfaces like 10/100Base-T MII Ethernet, General Purpose I/Os, time slot oriented PCM ports, and Univeral Serial Bus (USB).

The HARRIER-VT comes with a suite of software modules providing APIs, which will allow the user to effectively develop complete system solutions. The software can easily be adapted to various operating systems. Protocol software APIs allow easy integration of 3rd parties' higher layer protocol software (e.g. H.323, H.450.x, or x.GCP) as well as user's application software. Voice compression algorithms such as G.711,G.723.1 (including VAD and CNG), G.726, G.729A, G.729E, Acoustic Echo Cancellation (AEC), and DTMF can be provided in the form of linkable object code.

The evaluation kit EASY4261 will be available with engineering samples of HARRIER-VT. It may be used to exercise all functions of the HARRIER-VT and also to generate and analyze user defined IP traffic, or to capture incoming traffic for test and evaluation

Type	Package
PEB XXXX	F-TQFP-?

purposes. The VoIP reference kit will be provided with a complete VoIP application based on H.323 as a PCI plug-in card or stand-alone development board.

For software development and debug purposes a complete tool chain (incl. C/C++ compiler, linker, locater, profiles, assembler, and debugger) is provided by our tool partners such as Tasking, Greenhills and GNU.

3.1 Features

3.1.1 TriCore (CPU)

The TriCore is a 32-bit CISC/RISC/DSP with dual MAC and SIMD capability, optimized for real-time embedded systems. It implements a Harvard architecture with separate address and data busses for program and data memories. It has three pipelines: arithmetic, load/store, and loop control.

- High performance 80MHz CISC/RISC/DSP single core
- 32-bit load/store architecture
- Dual instruction issue
- 4 GigaByte address range (2^{32})
- General Purpose Register Set:
  - sixteen, 32-bit data registers
  - sixteen, 32-bit address registers
- Shadow registers for fast context switch
- Automatic context save-on-entry and restore-on-exit for:
  - subroutine
  - interrupt
  - trap
- Two memory protection register sets
- Data types:
  - boolean
  - integer with saturation
  - bit array
  - signed fraction
  - character
  - double word integers
  - signed integer
  - unsigned integer
  - IEEE-754 single precision floating point
- Dataformats:
  - bit
  - byte (8 bits)
  - half-word (16 bits)
  - word (32 bits)

- double-word (64 bits)
- Instruction types:
  - arithmetic
  - comparison
  - logical
  - shift
  - bit logical
  - bit field
  - packed data
  - address arithmetic
  - address comparison
  - MAC
  - coprocessor
  - branch
  - load/store
  - system
- Instruction formats:
  - l6 bit
  - 32 bit
- Addressing modes:
  - absolute
  - circular
  - bit reverse
  - base + offset
  - base + offset with pre- and post-update
- Multiply and Accumulate instructions (MAC)
  - dual 16 x 16
  - 16 x 32
  - 32 x 32
- Zero overhead loop
- Area = 5 mm^2 area
- Power = 1.5 mW/MHz @ 2.25V

3.1.2 Internal Bus (FPI)

The FPI bus is the high-speed internal bus that interconnects the TriCore core and internal peripherals. Please refer to "Peripheral Interconnect Bus, Version 3.3" for details. Implementation-specific features are listed here:

- Burst read and write of 2, 4 or 8 words
- No support for split transfers
- 32 bit address and data buses
- 1 master, 6 master/slaves and 15 slaves connected
- Limited single-master pipelining supported

3.1.3 Bus Control Unit (BCU)

- Handles bus arbitration between the on-chip FPI bus masters:
  - JTAG/OCDS port for debug support (TCU)
  - Peripheral Control Processor (PCP)
  - External Master Unit (XMU)
  - TriCore Program Memory Unit (PMU)
  - TriCore Data Memory Unit (DMU)
  - Transmit Data Management Unit (DMUT)
  - Receive Data Management Unit (DMUR)
- Acts as default bus slave when invalid/non-existent unit is addressed.
- Captures FPI bus error and timeout events for system debug.
- Implements algorithm to prevent request starvation due to bus domination by a high-priority master.
- Supports production testing of bus interconnect wires.

3.1.4 Data Memory Unit (DMU)

The HARRIER-VT contains a separate data memory unit (DMU). This unit consists of on-chip local memory, cache, and an interface to the FPI bus.

- Contains 32 kBytes of SRAM which may be configured in one of the following ways:
  - 16 kByte cache + 16 kByte scratch pad RAM (SPR)
  - 32 kByte scratch pad RAM (default after reset)
- Addressing:
  - data cache is not visible on the FPI bus.
  - scratch pad RAM is mapped in the first 16 kBytes (or 32 kBytes) of segment 13.
  - DMU configuration registers are mapped in the uppermost *256* bytes of segment 13.
- The Data cache has the following features:
  - two-way set associative
  - LRU replacement algorithm
  - line size = 128 bits
  - validity granularity = 1 valid bit per line
  - dirty bit granularity = 1 modified bit per 64-bit (2 per cache line)
  - flush, invalidate, flush + invalidate "address object" available
- The data cache objects can be individually flushed and/or invalidated to provide some support for cache coherency (to be handled by the programmer) through an instruction.
- The refill mechanism supports the following modes:
  - full refill of a cache line with burst-4 before data is accessed from/to CPU
  - no streaming

- The fetch accesses (interface to the CPU) support unaligned accesses (16-bit aligned), with a minimum penalty of one cycle for unaliged accesses crossing 2 lines (whether SPR or DCache lines).
- The data cache can not be bypassed for the cacheable segments. However, it always gets bypassed for the non-cacheable segments.
- Cache monitoring signals provided internally (not pins on the RIDER-B package):
  - cache access
  - cache hit

3.1.5 Program Memory Unit (PMU)

The HARRIER-VT contains a separate program memory unit (PMU). This unit consists of on-chip local memory, cache, and an interface to the FPI bus.

- Contains 16 kBytes of SRAM which may be configured in one of the following ways:
  - 8 kByte cache + 8 kByte scratch pad RAM (SPR)
  - 16 kByte cache
- Addressing:
  - instruction cache is not visible on the FPI bus.
  - scratch pad RAM is mapped in the lower half of segment 12.
  - PMU configuration registers are mapped in the uppermost 256 bytes of the lower half of segment 12.
- The instruction cache has the following features:
  - two-way set associative
  - LRU replacement algorithm
  - line size = 256 bits
  - validity granularity = 4 double-word per line
  - global invalidation support
  - not lockable
- The instruction cache can be globally invalidated to provide some support for cache coherency (to be handled by the programmer) through a write to a configuration register.
- The refill mechanism supports the following modes:
  - critical word first
  - no line wrap around
  - streaming to CPU
- The fetch accesses (interface to the CPU) support unaligned accesses (16-bit aligned), with a minimum penalty of one cycle for unaliged accesses crossing 2 lines (whether SPR or ICache lines).
- The instruction cache can be bypassed (default mode) to provide a direct fetch access from the CPU to the intemal on-chip bus resources.
- Cache monitoring signals provided internally (not pins on the RIDER-B package):
  - cache access
  - cache hit

3.1.6 Code Memory Unit (CMU)

The CMU is a second-level program memory which connects directly to the PMU.

- 32 kBytes of SRAM.
- Acts as a pipelined synchronous memory unit.
- Synchronous read and write behavior.
- 64-bit uni-directional input and output data buses.
- Burst read support (no wrap-around supported).
- Half-word (16-bit) write granularity.
- Independent and self contained BIST circuitry.
- All read/write accesses are naturally aligned on 64bit boundary.
- All read/write accesses have programmable 0 or 1 wait state.

3.1.7 External Bus Unit (EBU)

The External Bus Unit (EBU), also referred to as external bus controller, provides the external interface between the HARRIER-VT and the system devices. This interface has a demultiplexed 24-bit address and 32-bit data bus. The EBU's interface connects gluelessly to various types of memories and/or peripherals, including Intel-style peripherals (separate RD and WR signals), ROMs and EPROMs, Static RAMs, and synchronous DRAMs.

The EBU acts as a bridge between on-chip FPI masters and the external bus.

- External memory configuration support summary

Table 1

FPI Transaction	Demuxed Config	Muxed Config	SDRAM Config
Single (8-bit)	8, 16, 32-bit	8, 16, 32-bit	32-bit
Single(16-bit)	8, 16, 32-bit	16, 32-bit	32-bit
Single(32-bit)	8, 16, 32-bit	32-bit	32-bit
Burst(2 x 32-bit)	8, 16, 32-bit	32-bit	32-bit
Burst(4 x 32-bit)	8, 16, 32-bit	32-bit	32-bit
Burst(8 x 32-bit)	8, 16, 32-bit	32-bit	32-bit

- Supports classic SRAM, EPROM, ROM
- Supports synchronous DRAM:
  - It will be possible, using SDRAMs with $\overline{\text{CAS}}$ latency of two, to support bursts with zero wait-states between consecutive data of the burst at 60MHz or below.
  - must use 32-bit SDRAMs.
- Supports peripherals/ASICs with Intel-style interface
- No support for synchronous Flash memories

- Demultiplexed address and data bus mode
  - byte/half-word/word accesses from FPI to external devices of 8/16/ 32 bit data width in any combination supported.
- Multiplexed address and data bus mode
  - byte/halfword/word access from FPI to external device of the same data width as the FPI access or bigger data width supported
  - byte access to any size device possible
  - halfword access to 16 and 32 bit devices possible
  - word access to 32 bit devices possible
- 32-bit data bus
- Data widths of 8, 16, 32 bits
- 24-bit address bus
- 4 decoded user chip select outputs
- 2 decoded emulation chip select outputs (for overlay and emulation memories)
- Little endian operation (big endian is not supported)
- Programmable timing characteristics depending on address range: memory type, address generation, wait-states, etc.
- Byte write capability (four byte control signals)
- WAIT input for additional wait-state insertion
- External bus arbitration control with master/slave operation: HOLD, HLDA, BREQ

3.1.8 External Master Unit (XMU)

- Chip select input pin for access to internal FPI-Bus locations using an external bus master

3.1.9 Peripheral Control Processor (PCP)

The PCP performs most of the tasks that are normally done by a DMA controller and CPU interrupt service routines. It off-loads the CPU from most time critical interrupts with their inherent context switches and overhead, easing the implementation of systems based on operating systems. The PCP has no need for a kernel or task management. The PCP has limited computational capabilities and an architecture that efficiently supports DMA-type of bus transactions to/from arbitrary devices and memory addresses.

- Small, programmable, interrupt-driven microcontroller for data transfer and peripheral control. Includes instructions for DMA and bit handling.
- 40Mhz operation frequency
- PRAM 256 x 64 bit (parameter memory)
- CMEM 256 x 32 bit (code memory)
- 4 Gigabyte address range
- General purpose register set:
  - eight 32-bit registers for address, data, flags, and program counter

- Separate program and data/context memories
- Data types:
  - integer
  - bit manipulation
- Instruction Types
  - DMA (COPY)
  - arithmetic
  - logical
  - flow control/branch
  - load/store/exchange
  - shift
  - bit manipulation
- 16-bit Instruction Format
- Conditional execution of arithmetic, logical, jump
- Addressing Modes:
  - absolute
  - base + offset

3.1.10 Timers

3.1.10.1 Watchdog Timer (part of PWR unit)

The Harrier-VT Watchdog Timer provides a recovery mechanism from software or hardware failure.

3.1.10.2 System Timer (STM)

The System Timer is a high-precision, long-range 56-bit timer that provides a global system time for operating systems and other purposes.

- free running 56-bit system timer
- runs with CPU clock (e.g. 100MHz)
- timing range:2 56 counts (at 100MHz overflows after 22.5 years) - capable of timing the entire lifetime of the device
- no interrupt generated on overflow
- no interrupt unit
- no control bits and control registers
- starts counting after power-on reset
- not affected by any reset except power-on reset
- special capture register (SYSTIM7) is implemented (for 56 bits capture)
- optional 56-bit capture registers can be added to record external events
- scan test ready
- Gate Count:
- Area: 0.070 μm^2 (C9)
- Max. Speed: 66 MHz (C9)

3.1.10.3 General Purpose Timer Unit

The general-purpose timer unit, GPTU, consists of three basic 32-bit timers. The three timers are useful in timing events, counting events, and recording events. They can either run in standalonemode or be connected together to solve more complex tasks. They can also be split into several 8-bit or 16-bit timers.

- Gate Count:
- Area: 0.316 µm 2 (C9)
- Max Speed: 66 MHz (C9)

General Purpose Timers T0 and T1 (GPT0, GPT1)

- 32-bit up-counting timers/counters
- max. input frequency: SYSCLK / 2
- two input pins for count option
- dedicated 32-bit reload registers with automatic reload on overflow
- can be split into individual 8-, 16- or 24-bit timers with individual reload registers
- overflow signals can be selected to generate service requests, pin output signals, and T2 trigger events
- Gate count:

General Purpose Timer T2 (GPT2)

- 32-bit timer/counter
- up or down count option
- max. input frequency SYSCLK / 2
- operating modes:
  - timer,
  - counter,
  - quadrature counter
- options:
  - external start/stop, one-shot operation, timer clear on external event
  - direction control through software or external event
  - two 32-bit reload/capture registers
- reload modes:
  - reload on over/underflow, reload on external event: positive, negative, or both transitions
- capture modes:
  - capture on external event: positive, negative, or both transitions
  - capture and clear timer on external event: positive, negative, or both transitions
- can be split into two 16-bit blocks
- input pins freely assignable to timer count, reload, capture, etc. trigger functions, in addition to inputs from T0 and T1 overflows
- over-/underflow signals can be used to trigger T0/T1 and to toggle output pins

- T2 events freely assignable to the service request nodes

3.1.11 Serial Interfaces

3.1.11.1 Synchronous Serial Channel (SSC)

The High-Speed Synchronous Serial Interface, SSC0, supports full-duplex and half-duplex synchronous communication between the HARRIER-VT and other microcontrollers, microprocessors, or external peripherals. The serial clock signal can be generated by the SSC0 itself (master mode) or be received from an external master (slave mode). Data width, shift direction, clock polarity and phase are programmable, enabling communication with SPI-compatible devices.

- Master and slave mode operation
  - Full-duplex or half-duplex operation
- Flexible data format
  - Programmable number of data bits : 2 to 16 bit
  - Programmable shift direction : LSB or MSB shift first
  - Programmable clock polarity : idle low or high state for the shift clock
  - Programmable clock/data phase : data shift with leading or trailing edge of SCLK
- Baudrate generation from 12.5 MBaud to 190.7 Baud (@ 25 MHz module clock)
- Interrupt generation
  - on a transmitter empty condition
  - on a receiver full condition
  - on an error condition (receive, phase, baudrate, transmit error)
- Three pin interface
  - Flexible SSC pin configuration
- Scan test ready
- Gate Count: 4100 (C9-FPI)
- Area: 0.110 µm 2 (C9-FPI)
- Maximum speed: 66 MHz (C9-FPI)

3.1.11.2 Asynchronous Serial Communication Interface (ASC_P)

The Asynchronous/Synchronous Serial Interfaces ASC0 and ASC1 provide seriall communication between the HARRIER-VT and other microcontrollers, microprocessors or externall peripherals. The two ASCx interfraces are identical. The ASCx operate in either asynchronous or synchronous mode.

- Full duplex asynchronous operating modes
  - 8- or 9-bit data frames, LSB first
  - Parity bit generation/checking
  - One or two stop bits
  - Baudrate from 1.5625 MBaud to 0.37 Baud (@ 25 MHz clock f MOD)
  - Multiprocessor mode for automatic address/data byte detection

  - Loop-back capability
  - Support for IrDA data transmission/reception up to max. 115.2 KBaud
- Autobaud detection unit for asynchronous operating modes
  - Detection of standard baudrates1200, 2400, 4800, 9600, 19200, 38400, 57600, 115200, 230400 Baud
  - Detection of non-standard baudrates
  - Detection of asynchronous modes
  - 7 bit, even parity; 7 bit, odd parity;
  - 8 bit, even parity; 8 bit, odd parity; 8 bit, no parity
  - Automatic initialization of control bits and baudrate generator after detection
  - Detection of a serial two-byte ASCII character frame
- Half-duplex 8-bit synchronous operating mode
  - Baudrate from 3.125 MBaud to 318.5 Baud (@ 25 MHz Module clock)
- Double buffered transmitter/receiver
- Interrupt generation
  - on a transmitter buffer empty condition
  - on a transmit last bit of a frame condition
  - on a receiver buffer full condition
  - on an error condition (frame, parity, overrun error)
  - on the start and the end of a autobaud detection
- Gate Count: 5000 (C9-PD)
- Area:
- Maximum speed:

3.1.12 Parallel Port Module (PLP)

The on-chip GPTU, ASC0, ASC1 and SSC0 peripherals are connected to an 8-bit wide I/O port with flexible programming options for general purpose I/O or peripheral input/ output functions.

RIDER-B will have three 8-bit parallel ports.

- PortO
- Port 1
- Port2

3.1.13 Clock

- On-chip crystal oscillator circuit
  - Allows connection of external crystal, and supporting capacitors, via a two-pin interface (XTAL1,2).
  - Supports standard crystals from {min} to {max} MHz.
  - Permits external clock source driven through the XTAL1 pin of up to {max} MHz.
- On-chip Phase-Locked Loop (PLL)
  - Clock synthesis and frequency multiplication.

- Supports internal chip clock frequencies from {min} to {max} MHz.
- Two bypass modes for forwarding an external clock into the intemal clock system for chip testing.

3.1.14 Reset

- Several levels of chip reset are provided:
  - Power-on - resets entire chip
  - Hardware - used to synchronize board reset
  - Software - reset most of chip except key system resources (e.g. system timer).
  - Watchdog timer - recover from power down mode
- All flip-flops, except some special cases, are placed in a known state asynchronously. The reset tree is designed so that all parts of the chip will see the deassertion of reset within the same clock cycle.
- On-chip memories (i.e. SRAMs) are not reset and should be assumed to contain unknown values irnmediately after reset.

3.1.15 Power Management Unit (PWR)

The power management functions of the HARRIER-VT accommodate battery-powered devices. These functions provide long battery life and manage power loss. They also limit the peak power during turn on and reset of the HARRIER-VT in order to avoid an unnecessary Power Fault condition. There are four Power Modes: RUN, IDLE, SLEEP, and DEEP SLEEP.

- The clock tree is partitioned into several domains using clock gating cells. Each clock domain can be disabled for power saving. See "Figure 1. RIDER-B Functional Block Diagram" on page 12 for more.
- Clock domains are controlled in four ways:
  - Transparent (software invisible). Clock domain is shut off when determines that it is not needed. The clock is tumed on automatically when required. This method is used in each unit for the FPI clock.
  - Idle.
  - Sleep.
  - Chip Off.

3.1.16 Interrupt Systems

- *256* prioritization levels which can be used among two interrupt systems: TriCore and PCP.
- TriCore interrupt control unit (ICU) with a decentralized arbitration system.
- PCP interrupt control unit (PICU) also with a decentralized arbitration system.

3.1.17 Emulation and Debug Support

The HARRIER-VT provides a robust on-chip debug system. Its JTAG Interface conforms to IEEE 1149.1

- *Level 1:* JTAG port with FPI master capability for access to full intemal address space works in conjunction with OCDS_E, BRK_IN and BRK_OUT pins.
- *Level* 2: TriCore emulation monitoring unit (EMU) using a 16-pin interface for visibility and reconstruction of code execution. Program counter and BRK_OUT pins for monitoring program execution of the PCP.
- *Extended Level 2.*. not supported on RIDER-B.
- *Level 3:* not supported on RIDER-B.

3.1.18 Ethernet Port

3.1.18.1 Ethernet MAC

- two integrated MAC controller for 10Base-T and 100Base-FX/TX operation
- in half duplex mode, the controller implement the IEEE 802.3 Carrier Sense Multiple Access with Collision Detection (CSMA/CD) protocol.
- in full duplex mode, the controller implement the IEEE 802.3 MAC Control Layer and PAUSE operation for flow control.
- maskable notification of preset conditions
- implements counters for network management

The receive blocks accept incoming data from MII and perform the following operations:

- detection of SFD
- check CRC
- check minimum and maximum packet length
- lookup for address
- Address filtering capabilities for up to 5 MAC addresses
- Filtering for broadcast and multicast packets
- accept or reject packet.

Packets that have been accepted by the receive block are forwarded to the packet handler.

The transmit blocks forward the outgoing data from the packet handler to the MII and perform the following operations:

- generation of preamble and SFD
- generation of PADs
- generation of CRC
- generation of jam
- timer control for back-off after collision and for interpacket gap after transmission.

3.1.18.2 Ethernet Repeater

- provides two IEEE 802.3u compliant Media Independent Interfaces (MII) to LAN connection and external ethernet host computer.
- Supports 10Base-T and 100Base-TX/FX MII-based PHY devices
- Supports Full and Half Duplex Ethernet
- Includes MII Management Interface

3.1.19 PCM/HDLC Interface

Two SW configurable PCM Interfaces are provided consisting of a Port Interface, Time Slot Assigner and Protocol Machine.

OPEN issues:

Each PCM interface carries 32 time slots. A Frame Start signal will be provided for time slot zero. The accumulated maximum data rate will be 4.096Mbit/s.

- External CODECs or SLIC devices for the AFE
- Master and slave clock mode
- Single and double bit clock I/O
- Clock speeds configurable for 512kHz, 2,084kHz, 4,096kHz and 1,536kHz

3.1.19.1 Receive/Transmit Port Interfaces (XPI)

serial interface operating with gapped/non gapped serial clocks

- up to 32 channels with Nx8kbit/s (N=1...8) assigned for each PCM interface
- operation with strobed serial clocks **(future extension)**
- standard channelized mode (SCM) : 2 T1 (1.544MHz) / E1 (2.048 MHz) lines with chip internal framing function **(not supported)**
- alternate channelized mode (ACM) : 2 T1 (1.544MHz) / E1(2.048MHz) lines without chip internal framing function (E1/T1 arbitrarily portwise programmable)
- unchannelized mode (UCM) and mixed modes (SCM/UCM or ACM/UCM), UCM mode available for 2 ports
- serial data sampling/transmitting with rising/falling edge of clocks (ACM, SCM and UCM)
- ACM: sampling the receive/transmit sync pulses with rising/falling edge of clocks
- ACM: programmable TD/RD bitshift of +3/-4 bits relative to the sync pulse
- serial data multiplexable between normal and internal loop input
- supervision of frame synchronization conditions
- PCM-, HDLC-, framer- and interrupt-interface
- performance: serial clock up to 20MHz (port 0: 60MHz) ; SYSCLK up to 70 MHz
- full scan path for the SYSCLK domain

3.1.19.2 Time Slot Assigner (TSAT, TSAR)

- coordinates requests of 2 ports by built-in arbitration
- arbitration priority from low (port n = 0) to high (port n=1)
- channel programming by indirect access through TFPI (FPI Slave) Interface
- data output FIFO buffering for adjustment of further data processing
- programmable channel assignment per timeslot
- programmable mask per timeslot
- programmable timeslot inhibit flag per timeslot
- remote channelwise loop, one channel at a time (loop RD -> TSAR -> TSAT -> TD)
- remote portwise loop, one port at a time (loop RD --> TSAR --> TSAT --> TD)
- loop supported by a jitter attenuator, consisting of a 512 bit FIFO with slip function
- programmable "TMA1ST flag" (tmafirst flag) to identify the ' first' timeslot for TMA mode channels
- FPI target interface
- performance: SYSCLK up to 70 MHz
- scan path
- RAM (Timeslot Assigner Transmit Parameter Table: TATPT) built in selftest
- RAM (Timeslot Assigner Transmit Data Buffer: TATDB) built in selftest
- RAM (Timeslot Assigner Transmit Fifo: TATFIFO) built in selftest

3.1.19.3 Protocol Machine (PMT, PMR)

The two on chip HDLC Controllers are able to serve 64 logical channels which are assigned to independent PCM time slots.

- Protocols supported: HDLC, bit-synchronous PPP, octet-synchronous PPP, and Transparent modes
- full duplex operation
- Flexible scaling by dimensioning of context memory
- Per channel protocol configuration
- Configuration via Simplifed Microcontroller Interface (SMIF) registers
- Provides bit stuffing/deletion, flag detection/generation, CRC check/generation
- Aggregate throughput of 52.8 Mbit/s @ 33 MHz clock is possible

3.1.20 Transmit/Receive Data Management Unit (DMUT/DMUR)

- Controls linked lists for both transmit and receive direction arbitrarily assigned to the PCM/HDLC and Ethernet cores
- Provides independent internal frame buffers of 256 byte

3.1.21 USB Interface

The device itself may resume from suspend mode. The electrical interface and the protocol is compliant with USB specification 1.1.

- Full speed 12 Mbit/s USB device interface controller. All data transfers are initiated by the USB host
- USB Specificaton 1.1 compliant
- support of all USB device classes, including Communication Device, Audio and HID Class.
- 7 SW-configurable endpoints, in addition to the bi-directional control endpoint zero
- 3 configurations with 3 alternate settings and 8 interfaces supported
- Each non-control endpoint can be either isochronous (read-only), bulk or interrupt
- maximum packet length supported will be 1023 bytes.
- On-chip DMA support for up to 8 USB endpoints
- programmable DMA channel characteristics for each endpoint
- supports suspend and remote wake-up modes.
- Integrated USB transceiver

3.1.22 SW Package

Software package for VoIP reference kit includes

- Low level device driver software for on-chip communication modules
- Board support package for VxWorks
- TCP/IP stack with VoIP APIs
- APIs for H.323 and x.GCP
- Native TriCore voice codecs G.711, G.732.1 (incl. VAD and CNG), G.726, G.729A, and G.729E
- DTMF
- Acoustic echo cancellation (AEC)
- Real time T.38 fax including 14.4 bps data pump
- RTOS support for VxWorks, Nucleus+ and PSOS
- Small footprint micro kernel

3.1.23 Test ROM

3.1.24 Data-RAM with FPI interface

3.2 Typical Applications

3.2.1 IP LAN Phone

A low-cost IP LAN phone can be built with the HARRIER-VT. **Figure 1** shows the system block diagram of an IP LAN phone based on the HARRIER-VT.

All of the interfaces inside the dashed-line rectangle required for a stand-alone IP telephone, are provided on-chip. Other system components for the IP phone include the voice codec (Infineon's ARCOFI), two ethernet PHY devices, low cost ROM/SDRAM, display and keypad.

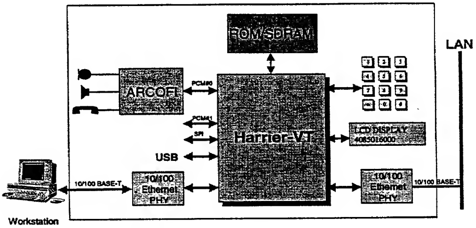


Figure 1 An IP phone based on the HARRIER-VT.

The IP phone is connected to the IP network via the enterprise LAN. A desktop workstation can also gain access to the LAN through the second ethernet port provided by the IP phone.

An optional USB device port is available for connecting USB devices to the IP phone for additional features.

The IrDA interface may connect PDA devices e.g. for phone directory service or email.

Real time operating system, VoIP protocols, voice codec SW, and ultilities software can be loaded into the external ROM or SDRAM.

3.2.2 Multi Channel VoIP Gateway

The HARRIER-VT is a scalable solution which can be configured to support up to 16 full duplex VoIP channels. The design is suitable for midsize residential gateways, multi channel Terminal-/LAN adapters or LAN linecard applications. **Figure 2** shows the functional block diagram of the basic implementation. The HARRIR-VT shipped in the 144-pin package will not support Ethernet and USB ports.

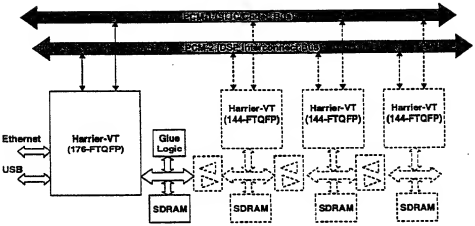


Figure 2 **A Multi-channel VoIP system**

In **Figure 2**, the HARRIER-VT (176-pin TQFP) is used as the VoIP master to the IP network providing the VoIP protocol stacks including RTP, UDP, IP and MAC access to the LAN.

Multiple HARRIER-VTs (with reduced package) can be connected to PCM-1highway which connects up to 16 SLIC/Codec devices, while the PCM-2 highway is used for data exchange between the HARRIER-VT used as a master and the N slave devices. In such a configuration, the programmer takes advantage of the HARRIER-VT as a DSP.

Using the same core in a system for both RISC and DSP, has the advantages of sharing the same kind of development tools, operating system, and application software.

When HARRIER-VT is used in DSP mode only, one HARRIER-DSP provides 100 DSP MIPS (at 80MHz) to handle 4 voice channels of full-duplex G.723.1 coding functions. The control communication and codec SW downloads between the master and the slave DSPs can be handled via the external processor bus or the SPI.

Figure 3 **Multi-channel VoIP system da<taflow**

3.2.3 Residential Gateway/VoIP Over Cable

The HARRIER-VT provides a highly integrated solution for cable modem VoIP applications. **Figure 4** shows an example of the cable modem with VoIP support.

Figure 4 **VoIP over cable using HARRIER-VT.**

For residential gateway applications the connection to the IP network can be provided by the HARRIER-VT as the cable modem controller.

- Up to 4 POTS phones or fax machines can be connected via the PCM interfaces.
- Depending on the VoIP coding algorithm implemented, up to four voice channels can be supported simutaneously.
- Two 10/100 Base-T ethernet ports (HUB) and a full-speed USB device interface is provided for workstation hook up.

The HARRIER-VT provides the DOCSIS-MAC with the MPEG payload and the flexibility required for implementing features such as Quality-of-Service, Service Enhancement & Feature Upgrades, and Control & Maintenance via the external bus interface. Besides this, the DSP capabilities will be used to implement the voice compression codecs on a per call basis.

When used as SDSL modem a SDSL-Transceiver connected to the PCM-Bus will be used instead of the external DOCSIS-MAC.

Gateway control protocols such as H.323, x.GCP or SIP will be implemented on the HARRIER-VT, since the RTOS is providing multi tasking capabilities.

3.2.4 Residential Gateway/Firewall

- Up to 4 POTS phones or fax machines can be connected via the PCM interfaces.
- One 10/100 Base-Tx/Fx port with MAC for workstation hook-up.
- One 10 Base-T port with MAC for workstation hook-up.
- Full-speed USB device interface is provided for workstation hook up.

In addition to the VoIP over cable application, encoding and decoding of the data packets is performed.

3.2.5 PCI Based VoIP System

A PCI VoIP adapter card can be built with the HARRIER-VT. **Figure 5** shows an example of a PCI adapter VoIP adapter card.

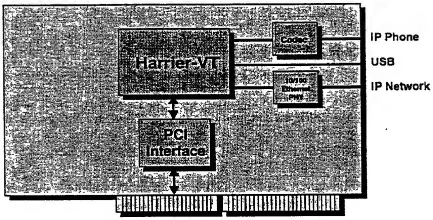


Figure 5 PCI based VoIP adapter card.

A basic PCI adapter card consists of the HARRIER-VT, Codecs, an ethernet PHY device, and a PCI bridge.

Without the PCI interface, the adapter card may have similar functions as a VoIP phone.

The PCI interface provides to the HARRIER-VT additional features such as sharing applications and data with the host system (therefore reducing external on-board memory) and other system modules.

Such a system is provided by Infineon as a VoIP reference card.

3.2.6 32bit USB-Controller

Device may be used as USB-IrDA Bridge

- 8 USB interfaces
- 7 + 1 USB endpoints
- isochronous traffic on USB
- 2 B-Channals via PCM

3.3 Target Criteria

3.3.1 Technology & Libraries

- C9N
- Four layer metalization
- Standard Cell Library:
  - Starlib (HL LIB)

- supplement to Starlib for TriCore including falling-edge DFFs (HL LIB)

- Pad Library: padlib_m3p30a_3v3? **tbd**
- Crystal Oscillator: osci_m3p80a_3v3 (PAX_ANC0 1P10M_JxN)? **tbd**
- Phase-Locked Loop: plLnderb_v1_1 (EZM, Villach, Austria)? **tbd**
- Register Files are implemented as fully synthesizable elements.

3.3.2 Operating Parameters

- 66 MHz (15 ns period) at typical process, worst temperature, worst voltage.
- I/O Supply. 3.3 V ±10% (3.0V to 3.6V), Low voltage TTL compatible outputs
- Core Logic: 2.5 V ±10% (2.25V to 2.75V)
- 110 °C junction temperature (70 °C ambient)
- Temperature range 0 °C to + 70 °C (optional -40 °C to +85 °C)
- ? Pin Quad Flat Package for F-TQFP-?, 0.5mm lead pitch **tbd**

5 Functional IC Description

5.1 Functional Overview

5.2 Block Diagram

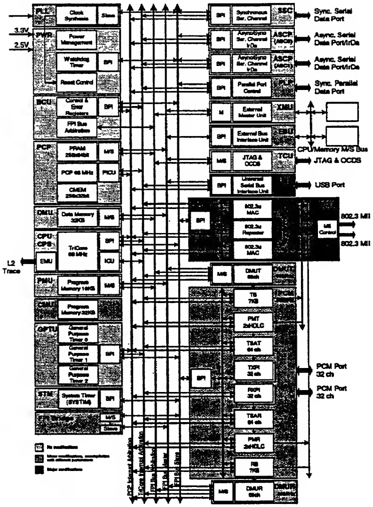


Figure 6 HARRIER-VT block diagram.

The HARRIER-VT block diagram is shown in **Figure 6.** It can be divided into four major functional blocks: TriCore processor core with memory and peripherals, ethernet MAC and repeater, USB and PCM/HDLC module.

5.3 Functional Blocks

5.3.1 Embedded TriCore processor core with memory and peripherals

This chapter only provides a general overview of the embedded TriCore and its peripherals. For detailed information please refer to the RIDER-B documentation.

The HARRIER-VT's embedded controller is based on Infineon's latest 32-bit TriCore architecture. The TriCore implements the computational capabilities of DSPs alongside the most optimized price/performance implementation of a RISC load/store architecture in a single core. It is well-suited for a wide variety of real-time oriented embedded control systems.

Other on-chip features of the HARRIER-VT's embedded controller include a 16-KByte instruction cache, a 32-KByte data cache, synchronous and asynchronous serial communication interfaces, a DMA/PCP, general-purpose timers, and sophisticated on-chip debug and emulation capabilities.

The embedded controller operates with a 80-MHz clock frequency. In DSP mode, the HARRIER-VT can deliver at least 100 DSP MIPS at 80 MHz.

5.3.1.1 TriCore Core CPU

The CPU is an Infineon TriCore superstar implementation. The 32-bit microcontroller/ DSP core contains two major pipelines that support integer and load/store operations and a third pipeline that supports optimized DSP loop operations. The integer pipeline executes basic integer instructions, bit operations, MAC instructions, divide, and conditional data jumps. The load/store pipeline executes load/store instructions, context operations, system instructions, address arithmetic and conditional/unconditional address instructions. Both the integer and load/store pipelines share a common fetch stage that can issue one instruction to each pipeline per cycle. The multiply-accumulate (MAC) instructions are executed in a dedicated two-stage MAC pipeline containing two 16x16 multipliers.

Loop optimizations are typically found within DSP applications and can be executed within the TriCore-1 pipeline using a dedicated loop hardware cache buffer (LCB). This buffer stores the location, target, and minimal set of information required to execute any repetitive loop within the dedicated loop pipeline. Unlike a normal Branch Target Buffer hit, the loop instruction itself is not fetched and is injected from the LCB into the loop pipeline during the last execute cycle.

Context operations associated with calls, returns, and interrupt entries are all supported via a 128-bit wide data bus between the register file and the local data memory. The core also contains dedicated hardware to optimize these operations, resulting in a reduction in the overall context save time to the scratchpad SRAM.

5.3.1.2 External Bus Interface Unit (EBU)

The external bus interface unit (EBU) provides a seamless interface to various external memories and peripherals, including EPROM, FLASHROM and SDRAM and "Intel-style interface" ASICs. Up to eight independent memory regions can be defined with different base address and length. Each region is associated with one chip select output and is programmable regarding bus protocol, wait states and byte ordering (little/big endian). The EBU also provides a demultiplexed 24-bit address and 32-bit data bus external interface. The data bus width can be programmed for 8-, 16-, or 32-bit wide accesses. Arbitration logic ensures that only one task is handled on the external bus at any one time and facilitates various operating modes including arbitration, external master mode, external slave mode, and stand-alone mode.

5.3.1.3 External Master Unit (XMU)

This unit acts as a bridge between external bus masters and the on-chip FPI bus.

5.3.1.4 On-Chip Memory

The on-chip memory of the embedded controller consists of a 16-KByte Program Memory Unit (PMU) and a 32-KByte Data Memory Unit (DMU). The PMU can be configured as 16-KByte instruction cache or 8-/8-KByte instruction cache/scratch SRAM. The DMU supports 32-KByte data cache or 16-/16-KByte data cache/scratch SRAM, Both memory blocks are supported with memory protection register sets to ensure both program and data integrity throughout system operation. Optional 32 KBytes CMU ROM for storing codeinstructions.

5.3.1.5 FPI Bus and Bus Control Unit (BCU)

The FPI Bus is used for on-chip interconnections. It connects the TriCore core with the peripherals including asynchronous and synchronous serial ports and external bus controller (EBU). Moreover, the communication modules like Packet Handler and Cell Processor are connected to the FPI Bus. The FPI Bus is a demultiplexed bus with 32 address bits and 32 data bits.

The BCU manages the arbitration between the devices residing on the FPI Bus.

5.3.1.6 General-Purpose Timer Unit (GPTU)

The GPTU consists of three basic 32-bit timers (T0, T1, and T2). T0 and T1 are functionally the same. The features of T0 and T1 are:

- 32-bit up-counting timer/counter
- two input pins for count option
- dedicated 32-bit reload registers with automatic reload on overflow
- can be split into individual 8-, 16-, or 24-bit timers with individual reload registers
- overflow signals can be selected to generate service requests, toggle output pins, and trigger T2 events

The features of T2 include:

- 32-bit timer/counter
- up or down count option
- timer, counter, or quadrature operating modes
- external start/stop, one-shot operation, two 32-bit reload/capture registers
- reloads on underflow/overflow
- reloads on external events: positive or negative transition or both
- captures on external event: positive or negative transition or both
- can be divided into two 16-bit timer blocks
- input pins assignable to timer count, reload, capture
- events freely assignable to service request modes

5.3.1.7 System Timer (STM).

The system timer provides a global time base for any operating system that might be ported onto the HARRIER-VT. The main features of this timer include:

- Free-running 56-bit timer system
- Operation with maximum clock frequency
- No interrupt on overflow
- Only a power-on reset affects this timer
- Timing range is 2^{56} counts

5.3.1.8 Watchdog Timer

The watchdog timer resets the HARRIER-VT when it is not serviced in time. In addition, the timer provides a program flow trace capability and double overflow detection. Program flow detection allows the programmer to set trace points in the program that must be reached and which are checked. Double overflow detection means that if a watchdog timer reset is immediately followed by another watchdog timer reset without initializing the watchdog timer first, the part is shut down completely because a save operation is no longer possible.

If the program takes an unpredicted flow, the watchdog timer cannot be reset because the password field in the control register will not contain the expected value.

5.3.1.9 Peripheral Control Processor (PCP)

The PCP is an I/O control processor that performs most of the tasks typically done by a dedicated DMA controller and CPU interrupt service routines. The PCP off-loads the CPU from time-critical interrupts, thus easing the implementation of systems based on operating systems. The architecture contains 1 KB of parameter SRAM and 1 KB of code memory. It is well-optimized to efficiently support DMA-type bus transactions to and from arbitrary devices and memory addresses. The PCP is also flexible enough to allow the implementation of a subset of the commands/instructions and operate as a simple DMA controller.

5.3.1.10 Serial Serial Channel (SSC)

The HARRIER-VT has a high-speed synchronous serial programmable interface wich connects to a number of codec devices or serial EPROMs.

5.3.1.11 Asynchronous Serial Communications Interface (ASCP)

The HARRIER-VT has two ASCPs that provides both full- and half-duplex asynchronous communications to other microcontrollers, microprocessors, or external peripherals. The baud rate for full-duplex asynchronous operation ranges from 1.5625 Mbaud to 0.37 baud at 25 MHz. The baud rate for half-duplex synchronous operation ranges from 3.125 Mbaud to 318.5 baud at 25 MHz. Parity, framing, and overrun error detection are provided for the asynchronous mode to ensure reliability of the data transfers. In a special synchronous mode, the ASC supports IrDA data transmissions up to 115.2 Kbaud with fixed or programmable IrDA pulse widths.

5.3.1.12 On-chip Phase Lock Loop (PLL)

The Clock Module divides a master clock frequency into lower frequencies required by different units within the HARRIER-VT for operation. The module contains an on-chip oscillator and PLL circuit, permitting the HARRIER-VT to operate on a low-frequency external clock while still providing maximum performance. The PLL also provides fail-save mechanisms that detect frequency deviations and enables the execution of emergency actions in case of an external clock failure. The HARRIER-VT's flexible clocking system minimizes power consumption and reduces EMI.

5.3.1.13 Resets and Interrupts

The HARRIER-VT performs the following reset functions:

- **External power-on or cold reset**
  An active-low reset pin configures the PLL and the clock circuitry through special configuration pins. It also sets an indication flag within the reset status register to inform the user about the reset type.

- **External hard or warm reset**
  An active-low reset pin serves as an external reset input and as a reset output. It can also connect to other external devices of the system in order to activate a total system hardware reset.
- **Software reset**
  The soft reset allows specific units and functions within the device to be reset without resetting the CPU and main system unit.
- **Watchdog timer reset**
  Wake-up reset from power-down mode

5.3.1.14 Debug Support

Emulation and debugging support for the HARRIER-VT provides system developers with full control over the execution of the target program and allows for simple and straightforward debug of the program through observation of its executions. In-circuit emulators, bus state analyzers, and software monitors are the more regularly used debugging tools. The HARRIER-VT supports internal bus visibility, program flow tracking, and breakpoint execution through its Test Controller Unit (TCU). The TCU implements the JTAG Interface and On-Chip Debug Support (OCDS) modules.

The OCDS connects to the FPI bus with emulation hooks to the core. This configuration allows priority communication between an external debugger and the internal system. It allows access to full internal address space. It also allows the user to utilize the JTAG port of the controller via a JTAG/COM port hardware connection and to communicate with the debugger. Using this method, the user can set simple breakpoint and trigger conditions because the JTAG communicates directly with the OCDS module. Reading and writing of data memory over the JTAG is also possible. The setting of breakpoints and the realization of these breakpoints are real-time and involve little or no slippage. Real-time run control is possible at the CPU frequency using this method.

5.3.2 Ethernet MAC and Repeater

The HARRIER-VT includes two 10/100Base-T ethernet Media Access Controllers (MAC).

In half duplex mode, the controller implements the IEEE 802.3 Carrier Sense Multiple Access with Collision Detection (CSMA/CD) protocol. In full duplex mode, the controller implements the IEEE 802.3 MAC Control Layer and PAUSE operation for flow control. The receive block accepts incoming data from MII and performs the following operations:

- detection of SFD
- check CRC
- check minimum and maximum packet length
- lookup for address
- accept or reject packet.

Packets that have been accepted by the receive block are forwarded to the packet handler.

The transmit block forwards the outgoing data from the packet handler to the MII andperforms the following operations:

- generation of preamble and SFD
- generation of PADs
- generation of CRC
- generation of jam
- timer control for back-off after collision and for interpacket gap after transmission.

A flow control block recognizes MAC control packets and supports the PAUSE operation for full duplex links. It also supports generation of PAUSE packets, and provides timers and counters for pause control.

The command and status registers control programmable options, including the enabling or disabling of signals which notify the system when pre-set conditions occur. The status registers hold information for error handling software, and the error counters accumulate statistical information for network management software.

An ethernet repeater provides two Media Independent Interface (MII) to LAN connection and external ethernet host computer. In VoIP application, the repeater LAN interface provides to HARRIER-VT the access point to ethernet network, and at the same time, it provides the routing path to ethernet host attached to the other MII interface.

5.3.3 USB Interface Controller

The HARRIER-VT provides a 12 Mbit/s (Full-Speed Mode) USB device Interface controller. The USB core provides maximum flexibility and performance, off-loading the CPU in such a manner that allows the user to implement value-added software features.

All USB data transfers are initiated by the USB host. The host is also suspending the device in order to save power and controls the remote wake-up. The device itself may resume from suspend mode. The electrical interface and the protocol is compliant with USB specification 1.1.

The HARRIER-VT provides eight endpoints: in addition to endpoint zero as the defaultcontrol Pipe, seven endpoints can be configured. Each of these endpoints can be of type isochronous, bulk or interrupt. Two configurations with alternate settings for multiple modes of device operation will be supported.

HARRIER-VT is designed to support f all USB device classes, including Communication Device, Audio and HID Class.

The maximum packet length supported will be 1023 bytes.

The DMA channel characteristics for each endpoint is programmable.

5.3.4 PCM/HDLC Module

The on chip HDLC Controller is able to serve two logical channels which are assigned to two independent PCM time slots.

In full duplex mode the controller provides HDLC mode as well as transparent mode.

The configuration of each logical channel is programmed individually. An aggregate bit rate of up to 8,192Mbit/s per direction is provided.

In HDLC mode the following tasks are performed:

- Zero Bit Insertion/Deletion
- Flag Detection/Insertion
- CRC Generation/Verification

In the transparent mode, the protocol controller performs fully transparent data transmission and reception without HDLC framing.

Two PCM interfaces are provided along with a time slot assigner which will be controlled by SW. Each PCM interface carries 32 time slots and a Frame Start signal will be provided for time slot zero. The maximum data rate will be 4.096Mbit/s.

The PCM module provides up to 16 independent 16-byte deep FIFOs for each direction.

Each PCM interface provides a single and double bit clock I/O according to master and slave mode operation.

5.3.5 Software Support

The overall software architecture is shown in **Figure 7** below. Besides the RTOS, such as Nucleus+, VxWorks and PSOS the HARRIER-VT comes with a set of well defined APIs for both VoIP related protocol stack software and user application specific APIs. In conjunction with native TriCore objects written in performance optimized assembly code for the voice codecs, including G.711, G.723.1, G.726, G.729A, and G.729E the programmer will have all necessary building block at hand for a specific VoIP application.

Figure 7 VoIP Software Modules for HARRIER-VT

Other applications such as real-time fax including V.17 data pump will be supported as well. Optional modules, such as acoustic echo cancellation (AEC), embedded HTTP server or TFTP will be available. V.90 softmodem software can be provided as a demo version for the HARRIER-VT. For a product version please contact your local Infineon sales representative.

5.4 Operating Modes

5.4.1 IP LAN Phone

The IP phone is plugged into an existing 10/100 Ethernet wall outlet. The workstation which may have occupied the outlet before, can be connected to the HARRIER-VT's second 10/100 Ethernet port.

- Both 10/100 Ethernet ports of HARRIER-VT are internally connected via a repeater.
- One 10/100 Ethernet port is connected to the IP network via the enterprise LAN.
- The second 10/100 Ethernet port enables LAN access for a desktop workstation.
- Up to 4 POTS phones or fax machines can be connected via the PCM interfaces.
- An external master may access on-chip-ressources for additional features via the external bus interface or the USB device port.
- The IrDA interface may connect PDA devices e.g. for phone directory service or email.

Figure 8 shows the IP LAN phone dataflow. Speech is encoded by an external voice encoder which is connected to the PCM ports. Encoded voicedata is transported via the receive branch of the PCM macro to the external memory. The TriCore reads this data, performs voice compression and writes it back to external memory. After that voice packets are read from external memory by the DMUT and forwarded to the IEEE 802.3u MAC. Then the data is fed into the LAN via the on-chip repeater.

In receive direction the IEEE 802.3u MAC listens to the ethernet traffic. Upon detection of the preconfigured MAC-Address, the MAC forwards the corresponding datapacket via RB, DMUR and EBU to the external memory. The TriCore reads the payload, performs decompressession and writes it back to external memory. After that the PCM coded voice is transported via the transmit branch of the PCM macro to external voice decoder connected to the PCM ports.

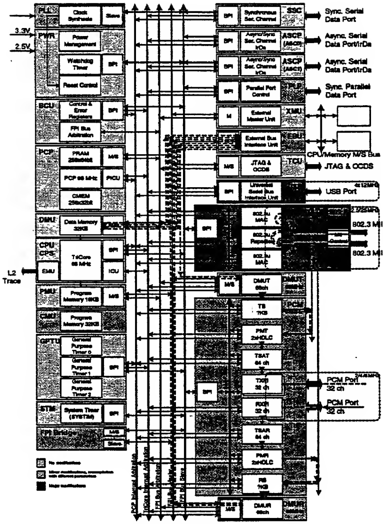


Figure 8 IP LAN Phone dataflow

5.4.2 Multi Channel VoIP Gateway

5.4.3 Residential Gateway/VoIP Over Cable

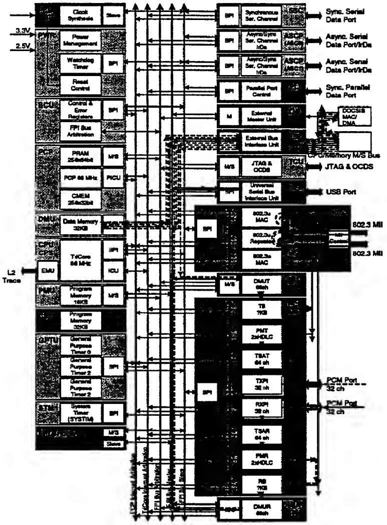


Figure 9 Residential Gateway/VoIP Over Cable dataflow

The dataflow for this application is shown in **Figure 9**. The external DOCSIS MAC copies received voice and data packets to the external memory. Voice packets are then read by the TriCore, decompressed and written back to external memory. After that the data is read by the transmit branch of the PCM module and forwarded to an external PCM encoder connected to the PCM port. Data packets are either moved from external memory to the USB port or via DMUT, TB and the IEE 802.3u MAC to the LAN.

In transmit direction encoded voice is forwarded via the receive branch of the PCM module to the external memory. The TriCore reads this data, does the compression and writes it back t the external memory. Data packets received from the MAC are forwarded via RB, DMUR and EBU to external memory. These voice and data packets are ready to be read by the external DOCIS MAC.

5.4.4 Residential Gateway/Firewall

Figure 10 shows the corresponding dataflow. It is basically the same as in the application above. After reception by the DOCSIS MAC data packets are read from external memory, encoded by the TriCore and written back to external memory before they are forwarded to the USB or ethernet interface. In transmit direction data packets written to external memory from USB or ethernet are encoded by the TriCore before transmittion via the DOCSIS MAC.

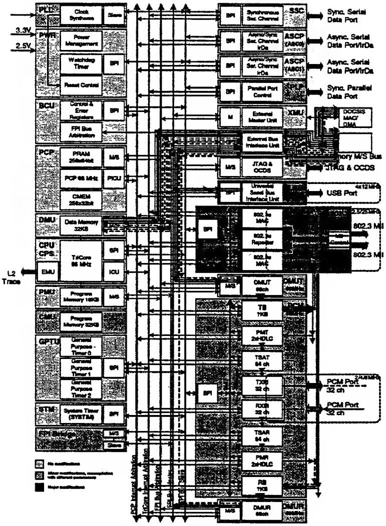


Figure 10 **Residential Gateway/Firewall dataflow**

5.4.5 PCI Based VoIP System

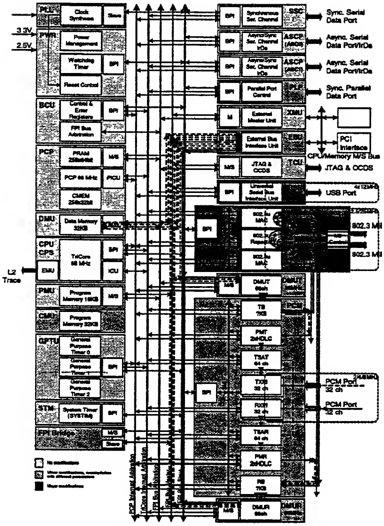


Figure 11 PCI Based VoIP System dataflow

5.4.6 32bit USB-Controller

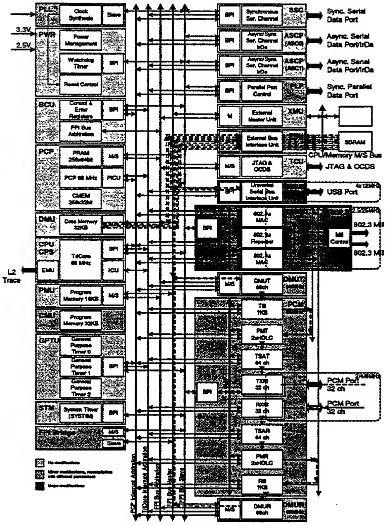


Figure 12 32bit USB-Controller dataflow

The PCM port is connected to an external voice codec. In transmit direction the voice samples are written through the receive branch of the PCM block to the external memory. The TriCore reads these packets, does voice compression and writes them back to the external memory. After that the voice packets ar ready to be read by the USB master connected to the USB port. In receive direction isochronous USB voice packets are written to the external memory. The packets are then read by the TriCore, decompressed and written back. Finally the voice samples are forwarded to the PCM port through the transmit branch of the PCM macro.

Appendix B

Title: Universal Serial Bus Device Controller 1.1

3 Overview

The Universal Serial Bus Device Controller (USBDC) interfaces the USB Function Device to the Universal Serial Bus. The USBDC handles the USB protocol and provides a Flexible Peripheral Interconnect Bus (FPI) interface on the local side. The internal Standard Microcontroller Interface (SMIF) provides a simple Read/Write protocol that allows an easy adaptation to other local bus protocols.

The USBDC Bus runs on a 12 MHz Clock which is extracted from an external 48 MHz clock on USB side. On the FPI side the USBDC is designed for up to 66 MHz. An internal FIFO is used for clock and data rate adaptation.

The USBDC can be configured in terms of number of configurations, interfaces and EndPoints depending on the device' s requirements.

3.1 Features

Some of the features of the USBDC are:

- Compliant with USB protocol revision 1.1.
- Support for both low speed and full speed devices.
- USB protocol handling.
- No microcontroller or firmware involved.
- USB device state handling.
- Clock and data recovery from USB.
- Bit stripping and bit stuffing functions.
- CRC5 checking, CRC16 generation and checking.
- Serial to parallel data conversion.
- Maintenance of data synchronization bits (DATA0/DATA1 toggle bits).

Type	Package
PEB XXXX	

- Supports up to three configurations with each configuration supporting four interfaces and each interface handling up to four alternate settings (Option to go up to eight alternate settings is also provided under certain circumstances).
- Programmable number of physical endpoints and supports up to 16 bi-directional logical endpoints.
- User configurable endpoint information.
- Understanding and decoding of standard USB commands to endpoint zero.
- Option to decode the GetDescriptor command or to pass the command onto the application bus.
- Support of Class/Vendor commands by passing the Setup transactions to the application bus.
- Support of string descriptors.
- USB clock: 12 MHz
- System clock: up to 70 MHz
- Number of gates:
- Area:
- Power consumption:
- Full scan path and BIST of on-chip RAMs for production test

3.2 Logic Symbol

3.3 Typical Applications

5 Functional Description

5.1 Functional Overview

The USB1 provides the USB functionality for the TriCore platform. This module provides a 12Mb/s full speed USB Vers. 1.1 compliant interface. It interfaces the USB transceivers on the one end and the Flexible Peripheral Interconnect bus (FPI) on the other end. Implemented as a slave device, it communicates with the Peripheral Control Processor (PCP) via side band signalling using interrupt signals.

The module provides the following functionality:

- USB device interface, compliant to USB specificatin, version 1.1.
- 12Mb/s full speed interface
- Support for audio, data and communication device classes.
- Two different configurations besides default configuration 0.
- One Bi-Directional control endpoint 0 and 15 SW-configurable unidirectional endpoints.
- Isochronous packet size up to 1023 bytes, control, bulk and interrupt packet size up to 64 bytes.

5.2 Block Diagram

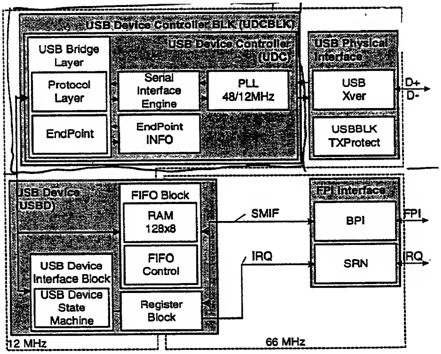


Figure 1 USB1 Block Diagram

5.3 Functional Blocks

The following paragraph gives an overview of the functions implemented in the macro's blocks.

For each block an indication about the changes to the existing UTAH's USBBLK is given.

5.3.1 USB Device Controller (UDC)

The USB Device Controller is a macro provided by Phoenix Technologies Ltd. Except of the configuration settings the Macro itself is not changed in this application. The following paragraph gives a overview of the UDC as it is described in the USB Device Controller User's Manual Rev. 2.0 [7] for detailed information see the referenced document.

The Universal Serial Bus Device Controller Core (UDC) interfaces the USB Function Device to the Universal Serial Bus. The UDC handles all the USB protocol and provides a simple Read/Write protocol on the Function Interface (Application Bus). The UDC and the Application Bus run on a 12 MHz Clock which is extracted from the 48 MHz clock provided by the User (FullSpeed Applications). The UDC does not have any FIFOs built in which helps the user to define his/her own FIFO requirements. The UDC is configurable in terms of number of configurations, interfaces and EndPoints depending on the Device' s requirements.

This block and its subblocks needs to be recompiled for the desired amount of Interfaces/endpoints. The necessay patch should be available at Infineon. All figures concerning number of Interfaces and endpoints need to be validated.

The main blocks in the USB Device controller are:

5.3.1.1 PLL Block

The PLL Block has a digital PLL built-in. The main functionality of the PLL Block is to extract the USB Clock and Data from the USB Cable. The input to the PLL Block comes from an external differential transceiver. The PLL runs on a 48 MHz clock for a full speed configuration. The PLL Block also generates a 12 MHz clock from the 48 MHz clock and can optionally supply to the SIE and UBL Blocks. The PLL identifies the single ended zero (SE0) signal on the USB and sends it to the SIE block. In the low speed configuration the PLL runs on a 6 MHz clock and generates a 1.5 MHz clock.

No modification necessary

5.3.1.2 Serial Interface Engine Block (SIE)

The second block is the Serial Interface Engine block (SIE) that does all the front end functions of the USB protocol such as SyncField identification, NRZI-NRZ conversion, token packet decoding, bit stripping, bit stuffing, NRZ-NRZI conversion, CRC5 checking and CRC16 generation and checking. The SIE also converts the serial packet to 8 bit parallel data. The SIE block has a one byte buffer built-in for buffering the data during data transmission and reception.

Recompilation may be necessary

5.3.1.3 USB Bridge Layer (UBL)

The third block is called the USB Bridge Layer (UBL) block that handles the error recovery mechanism during transactions while interfacing to the application. The UBL also decodes and handles all the standard control transfers addressed to EndPoint zero. The UBL passes all the Vendor/Class commands onto the Application Bus so that the application can decode the command and act there upon. This provides the flexibility of using the UDC core in multiple applications. The UBL supports up to a maximum of 3 configurations with each configuration having a maximum of 4 interfaces. Each interface can have up to 4(8) alternate settings. The UBL block has two sub blocks called the Protocol Layer (PL) and the EndPoint (EP) block.

Recompilation necessary

Protocol Layer (PL)

The PL Block controls the SIE Block by providing necessary Handshake signals to the SIE and talks to the Application Bus Logic. It also has the mechanism for error recovery if the Application Bus violates the data transfer protocol. The Application Bus protocol is explained later in the manual.

End Point (EP)

The EP block handles all the control transfers to EndPoint zero. The EP block decodes and responds to all the USB standard commands and passes the USB Class/Vendor commands to the Application Bus. The EP block maintains the buffer for device address, buffer for storing the present active configuration, and the logic to determine the present state of the device.

5.3.1.4 EndPoint Information Block (EPINFO)

This is a configurable block in the UDC Core that can be configured by the user by using the UDC Compiler (RapidScript). This enables the UDC to be configured for different applications and tailors to the needs and requirements of different applications. The EPINFO block maintains the buffer (registers) that stores the information about all the EndPoints the device supports. This also stores the information about the size of configuration and all string descriptors that are supported in the UDC core. The information about the current EndPoint is multiplexed from these buffers and is provided to the PL block which inturn will control the SIE block based on this information. The EPINFO also has the DATA0/DATA1 synchronization bits for each bidirectional EndPoint the UDC supports. The EPINFO also has the EndPtStalled Bit for each of the supported logical EndPoint to indicate the stalled status of the EndPoint. The EPINFO supports up to 16 active bidirectional Logical EndPoints. The EndPoint Number ranges from 0 to 15. The number of Physical EndPoints is programmable.

Recompilation necessary

5.3.2 USB Device Controller Block (UDCBLK)

The UDCBLK is a wrapper around the UDC, where the following signals are connected:

- pllreset and reset: Connected to x_res_n even if the UDC specification says that the pllreset must be taken away at least two clock cycles before the reset is taken away. This is possible because the synchronous reset into the UDC was modified into an asynchronous reset of all the registers.
- dev_clk_gated is simply connected to udc_clk - in the UDC manual they say there is a possibility to gate this clock signal with gate_dev_clk but we don't use this possibility

because the chip itself has the capability of tuming off the clock for the entire UDC in suspend mode.

Changes that were made to the UDC:

- All resets of registers which can be reset changed from synchronous to asynchronous reset.
- Added a asynchronous reset to all registers except in FIFO and DPLL modules.
- In scan mode switching of the clocks in the module DPLL.
- In scan mode turning off the asynchronous set of a register in the SUSP_RES module.
- Added an input DEV_Milisec for generating an millisecond pulse in the SUSP_RES module by writing into a dedicated register in order to reduce the simulation time of a suspend/resume test. A write to this register fakes the assignment of a millisecond pulse which is usually generated by a counter.

Slight modifications due to extended pinout of the UDC

5.3.3 USB Physical Interface (UPHY)

This block contains the circuitry necessary to fulllfill the electrical specification of the USB. It contains the USB Transceiver (USBXVER) and the USB transceiver protection (USBBLK TXPROTECT). For details please see the USB specificatin, Chapter 8 [1)]

No modifications necessary

5.3.3.1 USB Transceiver (USBXVER)

This block contains the differential transmitter that is used for driving the USB data signal onto the USB cable and the differential receiver used for receiving the data from the cable.

Migration from C7 (UTAH) to C9 (Harrier-VT)

5.3.3.2 USBBLK TXPROTECT

This block is for protecting the USB transceiver from shortcuts on DPLS and DMNS. Th USB specification[1)] requires, that the transceiver which is connected to DPLS and DMNS (which is at 5V) must resist a short-cut of DPLS and DMNS to VDD or GND for a long time. In order to prevent the transceiver from being damaged, it will be turned off while a shortcut is detected.

No modifications necessary

5.3.4 Flexible Peripheral Interconnect Interface (FPIIF)

The Flexible Peripheral Interconnect Interface (FPIIF) contains a generic Bus Peripheral Interface (BPI) that interconnects the Flexible Peripheral Interconnect bus (FPI) and the Slave Module Interface (SMIF). It also contains a Service Request Node (SRN) for interrupt arbitration to the external PCP.

5.3.4.1 Bus Peripheral Interface (BPI)

Please see the Design Documentation BPI for FPI, Version 3.10 [10] for a brief description of the module's functionality.

Generic Macro has to adopted.

5.3.4.2 Service Request Node (SRN)

Please see the TriCore Architecture Manual [12] and the Microcontrollers ApNote AP3222 [13] for a brief description of the module's functionality

Generic Macro has to be adopted.

5.3.5 USB Device (USBD)

This block interfaces the USB Device Controller (namely UDCBLK) via the UDC's application interface and interfaces the FPI Interface block via the SMIF register interface. It consists of the following blocks:

Some blocks need to be redesigned

5.3.5.1 USB Device Interface Block (UDBLK)

The USB Device Interface block consists basically of the USB Device State Machine (UDSM) and the multiplexing logic needed to multiplex the current active transmit/ receive EndPoint channel with the UDSM. The multiplexing is done transparent to the UDSM, i.e the UDSM doesn't even know the existence of the 8 transmit/receive EndPoint Channels, it is designed to operate as if there is only one transmit channel from which it reads transmit data and only one receive channel to which it writes data that is received from the UDC. The MUX logic takes care of routing the data to the appropriate transmit or receive channel.

Adaptation to the actual amount of interfaces/endpoints

5.3.5.2 Register Block (RBLK)

The USBD Register Block contains the registers that are defined for the USBD block. This block also contains the RxByteCount[15:0] registers for all the 8 channels. These RxByteCount registers belong logically to the corresponding EndPoint Block of the UDC but they are physially implemented in this block.

Located in the USB clock domain the Register Block has to do the clock rate adaptation for the input signals from the FPI interface block.

Additionally this block contains all the miscellaneous logic that doesn't belong to any other block.

5.3.5.3 FIFO Block (FIFOB)

The FIFO Block is used for clock rate adaptation and data buffering. As the UDC access to memory needs to be granted within 4 clock cycles a local storage is necessary. In addition this block transforms the 8bit UDC application bus to the 32bit SMIF.

Located in the USB clock domain, the FIFO block has to do the clock rate adaptation for the input signals from the FPI interface block.

Logically, the FIFO block contains a transmit FIFO (TXFIFO) or a receive FIFO (RXFIFO) for each EndPoint. Physically, theses FIFOs are implemented by 128x8bit (dualported?) RAM (16 endpoints with 8 byte each) and the corresponding control machine.

New design block

FIFO Control (FIFOC)

The FIFO control block is responsible for maintaining the buffer start and end pointers for each channel independantly. For this it implements one controller for each physical port of the FIFO. As only one endpoint can be active at a given point of time, the controllers load the associated pointers from an internal register array or memory.

The controller on the UDC-side has to guarantee 4-clockcycle access for the UDC. Therefore, UDC transfers have precedence over SMIF transfers. If necessary, pending transfers on the SMIF side have to be stalled, resulting in waitstates on the FPI bus.

New design block

5.4 Operating Modes

Appendix C

Title:
Flexible Peripherals Interconnect Bus Version 3.2

1.1 FPI-Bus Features

The FPI-Bus is designed with requirements of high-performance systems in mind. The features are:

- Core independent
- Multi-master capability (up to 16 masters)
- Demultiplexed operation
- Clock synchronous
- Peak transfer rate of up to 800 MBytes/s (@ 100 MHz bus clock)
- Address and data bus scalable (address bus up to 32 bits, data bus up to 64 bit)
- 8-/16-/32- and 64 bit data transfers
- Broad range of transfer types from single to multiple data transfers
- Split transaction support for agents with long response time
- Burst transfer capability
- EMI and power consumption minimized

FPI-Bus does **not** provide:

- Cache coherency support
- Broadcasts
- Dynamic bus sizing
- Unaligned data accesses

1.2 FPI-Bus Overview

FPI-Bus is an on-chip bus to be used in modular, highly integrated microprocessors and microcontrollers (*systems-on-chips*). FPI-Bus is designed for memory mapped data transfers between its bus agents. Bus agents are on-chip function blocks (modules), equipped with a FPI-Bus interface and connected via FPI-Bus signals. A FPI-Bus agent acts as a FPI-Bus master when it initiates data read or data write operations once bus ownership has been granted to the agent. A FPI-Bus agent which is addressed at a FPI-Bus operation acts as a FPI-Bus slave when it performs the requested data read or write operation. Typical masters are processor modules, coprocessors, DMA controller or the external bus interface. Typical slaves are on-chip peripherals (**Figure 1-1**).

To increase the availability of the bus, read transaction may be split into two independent transfers: first the transfer request with some additional information on data type and block size and later the transfer of the requested data from the slave to the master. For the second part the slave will become master on the bus and the previous master the slave. Agents which support this type of transfers are called master-slave agents (refer to 3.4.5"Master-Slave Agents,").
Memory modules may be implemented as pure slaves (normal case) or as master slave modules, depending on their speed.
Other examples for FPI-Bus agents that have to provide master as well as a slave functionality are

coprocessors. A coprocessor may need to be initialized by the processor before it can be started. To initialize, the processor has to write values into the coprocessor's registers via the FPI-Bus interface. In this case the FPI-Bus interface of the coprocessor operates as a slave. When the coprocessor is allowed to run, it reads from or writes to memory or communication interfaces by its own. The FPI-Bus interface now acts as a master.

To operate, FPI-Bus requires an additional bus controller which ensures the bus protocol. It could do e.g. time-out and slave access control or protocol handling for all simple slave devices.
The bus controller functionality will be implementation dependent due to more or less centralized control approach.

The FPI-Bus protocol is oriented towards fast FPI-Bus agent accesses as well as to a high transfer bandwidth. A low-overhead protocol guarantees short response times at FPI-Bus accesses which are needed for time-critical applications. Multiple data transfers allow FPI-Bus to operate at a high bandwidth.

There are three types of agents possible on the FPI-Bus (see **Figure 1-1**):

- Master agents which can initiate and control transactions.
- Slave Agents, which only support simple read and write of registers and are not dealing with the bus protocol. This has to be done e.g. by a bus controller and
- Master-Slave Agents, which will support advanced features like split read transfer support and error handling. Depending on the type of transaction these agents may act as master or slave or both.

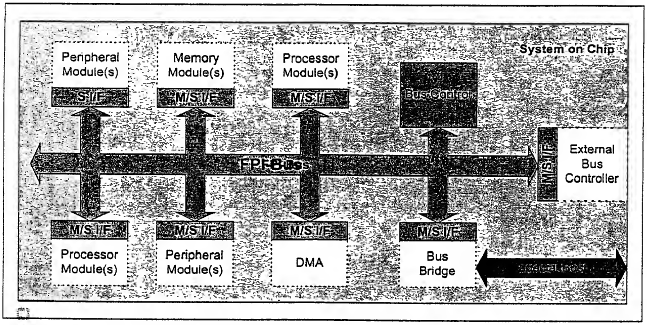


Figure 1-1: Examples for Modules of a FPI-Bus system

The FPI-Bus is designed to be flexible enough to support different processor cores, provide high data bandwidth and high availability. It can interconnect all kinds of internal and external peripherals and memories to different active bus agents like CPUs, DMA/PEC controllers and coprocessors. Special care was taken to reduce power consumption and EMI.

This is achieved by providing:

- scalability of address and data bus
- flexible bus protocol, only subsets may be implemented or optional extensions defined can be incorporated
- support for split read transfers (request of transfer and transfer of the data may be separated)
- bus signal hold circuitries for most bus signals (to prevent undefined state on undriven bus lines)

Figure 1-2 shows an example for a simple FPI-Bus implementation.

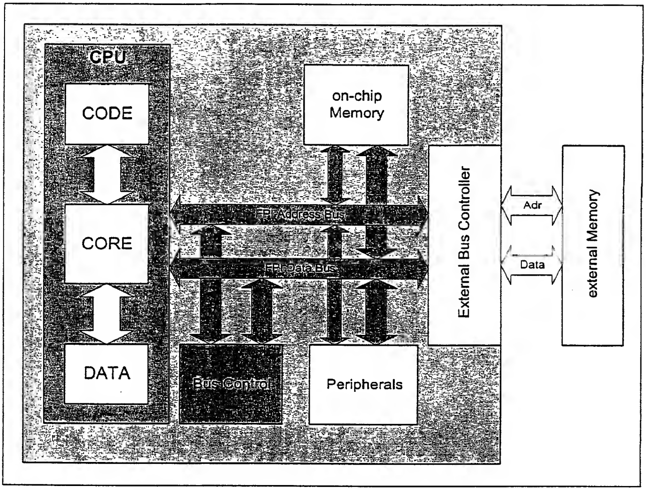


Figure 1-2: Example of a simple FPI Implementation

Although the FPI-Bus does not provide a cache coherency support, agents on the FPI-Bus may include caches. The only restriction is, that all accesses to cached areas have to go through the cache and must not bypass it (see Figure 1-3).

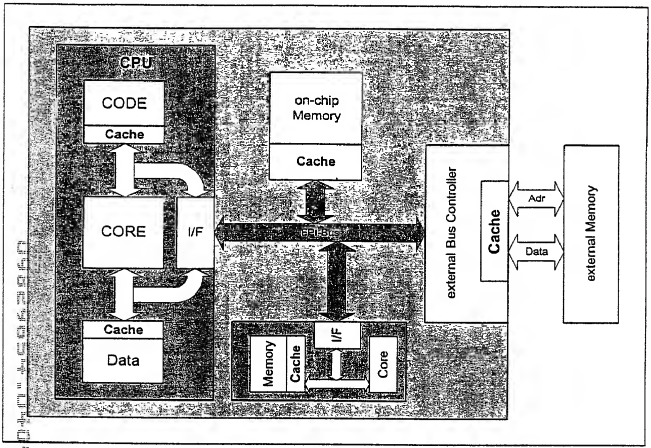


Figure 1-3: Example of caches in a FPI-Bus based system

NOTE:
In case of other bus masters sitting on the external memory bus, cache coherency on this bus has to be ensured by suitable design of this bus.

1.3 Scope of FPI-Bus Specification

The FPI-Bus specification covers the FPI-Bus protocol. It describes which set of transfers and signals at least have to be supported by FPI-Bus agents. It also lists options which may be supported by a FPI-Bus agent.

NOTE:
The FPI-Bus specification does not regulate implementation issues. In particular, it does not specify the layout of a FPI-Bus interface or define electrical parameters. This has to be specified separately depending for each implementation of the FPI-Bus.

1.4 Data Bus Width

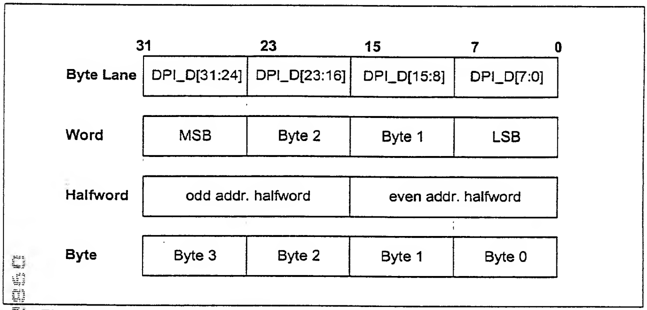
The FPI-Bus supports data bus widths of 16, 32 and 64 bits. For simplicity the explanations and examples in this document focus on a 32-bit implementation of the bus. 16-bit or 64-bit implementations easily can be derived by changing the data bus width, the basic bus protocol will not be effected.

NOTE:
Any building of multiple accesses e.g. to transfer 64 bit data on a 32-bit data bus implementation must be done by master.

1.5 Address Bus Width

The FPI-Bus supports address bus widths of 16 to 32 bits. For simplicity the explanations and examples in this document focus on a 32-bit implementation of the bus.

For simplicity the examples show the alignment for up to 32-bit data. For 64-bit data they have to be expanded accordingly.



	31	23	15	7 0
Byte Lane	DPI_D[31:24]	DPI_D[23:16]	DPI_D[15:8]	DPI_D[7:0]
Word	MSB	Byte 2	Byte 1	LSB
Halfword	odd addr. halfword		even addr. halfword	
Byte	Byte 3	Byte 2	Byte 1	Byte 0

Figure 2-2: Alignment of bytes, halfword and words on FPI-Bus data lines (32-Bit implementation)

3.1 Overview of FPI-Bus Signals

The following table lists all FPI-Bus signals and some additional information about these lines. A detailed description of all signals is given in 3.2"Description of FPI-Bus Signals," on page 26.

Table 3-1: FPI-Bus Signals

Signal Name	Master	Slave	Bus Control[a]	Comments
FPI_RES_N[1:0]	I	I	I	Reset Signals
FPI_CLK	I	I	I	Bus Clock
FPI_REQ_x_N	O	-	I	Master x Request line
FPI_RES_REQ_x_N	O	-	I	Master x Split Response Request
FPI_GNT_x_N	I	-	O	Master x Grant line
FPI_BUSY_x_N	O	-	I	Pending Split Response
FPI_LOCK_x_N	O	-	I	Master x Lock line
FPI_RD_N	O	I	-	Read Control
FPI_WR_N	O	I	-	Write Control
FPI_ABORT_N	O	I	-	Abort an already started transaction
FPI_OPC[3:0]	O	I	-	Operation code
FPI_A[31:0]	O	I	I	Address bus, width as required
FPI_D[63:32] FPI_D[31:0] FPI_D[15:0]	I/O	I/O	-	Data bus
FPI_TAG[3:0]	O	I		Tag Bus (used to transfer the master ID for Split Block Transfers)
FPI_ACK[1:0]	I	O	I	Slave response code
FPI_RDY	I	O	I	Slave on data bus will be able to finish transaction in current clock cycle
FPI_SEL_y_N	-	I	O	Slave select
FPI_TOUT	I	I	O	Time-out signal
FPI_SVM	O	I	-	Supervisor Mode

Table 3-1: FPI-Bus Signals

Signal Name	Master	Slave	Bus Control[a]	Comments
FPI_NO_SPLIT_N	-	I	(O)	No Split Response

[a] including bus arbiter and address decoder

The ending "_N" means that the signal is low active (V_{SS} level: signal is active, V_{CC} level: signal is not active).

3.2 Description of FPI-Bus Signals

3.2.1 System Signals

FPI_RESET_N[1],[0] (Bus Reset)

Reset forces all bus masters and slaves interfaces into idle state. In addition the peripherals can be reset. At least one reset line **FPI_RESET_N[0]** is needed. The second line **FPI_RESET_N[1]** is recommended to enable the distinction of interface reset and peripheral reset. The following table shows the coding of the reset lines (recommended reset states are print italic).

Table 3-2: Reset Encoding

FPI_RES_N[1]	FPI_RES_N[0]	Reset State
0	0	Complete Reset of all Bus Interfaces and all Peripherals
1	1	No reset
0	*1*	*Reset of all Peripherals (int. Regs. -> Reset Value)*
1	*0*	*Reset of all Bus Interfaces (no changes to int. Regs.)*

During the power-up phase **Reset** can be asynchronous, during normal operation it shall be activated synchronously to bus clock. **Reset** is always deactivated clock synchronous (with rising edge).

FPI_CLK (Bus Clock)

Bus cycle timing is referenced to the leading edges of the bus clock (**FPI_CLK**). The only exception might be the bus arbitration to enable single cycle master switch (refer to “Bus Arbitration,” on page 38).

3.2.2 Bus Ownership Control Signals

All Bus Ownership Control Signals are dedicated point-to-point lines.

FPI_REQ_x_N (Bus Request)

FPI_REQ_x_N shall be driven by a master early in a bus cycle to request bus ownership from bus control for the following cycle.
All masters with active **request** lines take place in the arbitration round for the following bus cycle. If, due to a default grant, bus ownership has already been given to the master in the previous bus cycle, the master can indicates via **FPI_REQ_x_N** whether it needs the bus for one more cycle (than it takes place in the current arbitration round) or not.

Each master has a separate **FPI_REQ_x_N** line to the bus arbiter.
Therefore the number of **request** lines depends on the number of masters in the system.

A granted master has to release **FPI_REQ_x_N** immediately at the beginning of the address cycle if no further bus requests shall be issued.

NOTE:
Priority control of **request** lines is not part of this document but implementation dependent.

FPI_LOCK_x_N (Lock Bus Request)

A master that requests the bus for a certain number of consecutive, non-interruptible (chained) transfers has to issue the **Lock** line in parallel with the **Request** line. If this master is granted the bus arbiter must not perform any arbitration round until this master releases its **Lock** again.
The **Lock** line is recommended for masters that perform e.g. read-modify-write transactions.

This leads to the following kinds of requests:

Table 3-3: Locked Bus Request Encoding

FPI_REQ_x_N	FPI_LOCK_x_N	Request
0	0	Locked Bus Request
0	1	Bus Request (unlocked)
1	x	no Request

FPI_LOCK_x_N can be released without releasing the **FPI_REQ_x_N**. In this case normal arbitration goes on.

The use of locked requests is implementation specific. One should take care to avoid deadlock situation that might occur if the number of locked bus transactions is too high.

FPI_RES_REQ_x_N

A master-slave interface shall use this signal to request the bus for a pending split response. This enabled the arbiter to distinguish between normal request and pending response request.
In systems without split transactions **FPI_RES_REQ_x_N** and **FPI_REQ_x_N** shall be ored together either inside the interface or inside the arbiter (recommended).
This line is a point-to-point line from a split capable interface to the bus arbiter.

FPI_GNT_x_N (Bus Grant)

At the end of an arbitration round the bus arbiter drive one grant line (**FPI_GNT_x_N**) active to indicate to a master that its request for bus ownership has been accepted.

NOTE:
If a master did not request the bus but is granted it has to perform idle cycles.

Only one **FPI_GNT_x_N** may be active at the end of a bus cycle.

❑ **FPI_GNT_x_N** = 1: Master does not get bus ownership.

❑ **FPI_GNT_x_N** = 0: Master gets bus ownership.

Definition: A master is granted in cycle N if its dedicated **FPI_GNT_N** line and **FPI_RDY** was active in cycle N-1. It owns the data cycle starting in cycle N+1 ending with the cycle in which the transaction is terminated by activating **FPI_RDY.**

A granted master shall start operation by performing

❑ an address cycle or

❑ an idle cycle.

Earliest during this cycle **FPI_GNT_x_N** may be released again.

Each master has a separate **FPI_GNT_x_N** line from bus control.
Therefore the number of grant lines depends on the number of masters in the system.

For further details on arbitration refer to Section "Bus Arbitration," on page 38.

FPI_BUSY_x_N (optional)

A master-slave interface gathering data for a pending split response shall drive this line active. This informs the arbiter that there is a response pending.
This line is a point-to-point line from a split capable interface to the bus arbiter.

NOTE:
This line is optional and must be inplemented only in slave agents that support split transactions.

FPI_SEL_x_N (Slave Select)

This signal is a kind of chip select for the slave interface. Each slave interface has one dedicated **FPI_SEL_N** input.

Definition: A slave is selected in cycle N if its dedicated **FPI_SEL_N** line and **FPI_RDY** was active in cycle N-1. This selection is valid (for one or more cycles) until the transaction is terminated either by activating **FPI_RDY** (by the slave) or by activating **FPI_TOUT** by Bus Control.

Only the selected slave interface is active. All others have to remain in idle mode.

NOTE:
The select signal generation is not part of this specification due to the various number of different possibilities and system requirements. An example for an address decoder to generate these select signals is given in Appendix 6.5"Address Decoder (FPI_SEL_N generation),".

3.2.3 Bus Operation Signals

All bus operation signals are bussed lines which shall be driven in every active bus cycle. To avoid floating of not driven lines in case of idle cycles or bus error they will be protected with bus hold circuitries.

These bus hold circuitries are not used for bus functionality but only for power savings and lifetime extension.

FPI_READ_N, FPI_WRITE_N (Read/Write)

The read/write bus lines are driven by the granted master in address cycle to inform the selected slave that it has to provide resp. has to latch valid data in the data cycle.
(An opcode and the slave address is issued in parallel.)

Table 3-4: Read/Write Enconding

FPI_RD_N	FPI_WR_N	Description
0	0	Read-Modify-Write
0	1	Read Operation
1	0	Write Operation
1	1	NOP

Read-Modify-Write (RMW) Accesses

The FPI provides a special RMW indication for bit protection. This indication can be used by the peripheral to avoid (postpone) any internal write accesses to the addressed register during the read-modify-write.
The first read access of a RMW shall be done with read and write line active (see Figure 4-1). After the second FPI-access - a write - the internal write access shall be allowed again.

NOTE:
RMW is done in at least two cycles which shall be locked. In the first one data is read, in the second one the modified data is written back. In between no other bus agent can gain control of the bus.
A RMW accessed slave must not acknowledge with SPLIT.

If the master needs additional cycles to modify the data it shall perform idle cycles while the bus is still locked.

Idle Cycles

Both lines **FPI_RD_N** and **FPI_WR_N** shall be driven inactive.

FPI_ABORT_N (Abort FPI Transaction)

An already started transaction can be cancelled with this signal. **FPI_ABORT_N** must be issued in the (first) data cycle to abort the current transaction. (refer to Chapter 4.6"Error Handling," on page 74).

NOTE:
Each slave with side-effect registers shall support abort functionality. That means
a) read access => the slave should restore registers with side effects
b) write access => the slave should not take the data.

FPI_A[31:0], FPI_A[63:32] (FPI Address Bus)

Address bus lines are driven by the granted master in the address cycle. The address identifies the slave which is addressed by the bus operation. Based on this address the slave select (**FPI_SEL_x_N**) shall be generated.

NOTE:
The number of address lines to be implemented depends on system address space requirements and the address map of the implementation.

FPI_D[15:0], FPI_D[31:0], FPI_D[63:0] (FPI Data Bus)

Data bus lines are driven by the master or the slave (depending on read or write) during data cycle. The data is taken at the end of the (last) data cycle which is indicated by an active **FPI_RDY** signal.

NOTE:
The number of data lines to be implemented (16, 32 or 64) depends on the system implementation.

FPI_TAG[3:0] (FPI Tag Bus)

To enable split transfers an ID of the requester has to be transmitted on the Tag Bus. This ID is used to address the right bus agent for the split response.
Due to this scheme any master can have only one split transfer pending.

NOTE:
Due to the bus width the number of possible masters in an implementation is limited to 16.

FPI_SVM (FPI Supervisor Mode)

Distinction whether a master accessing the bus is running in supervisor mode or in user mode. This signal has to be issued in the address cycle.

- FPI_SVM =1: master is in supervisor mode
- FPI_SVM =0: master is in user mode

A slave that only supports supervisor mode for a given access, and is not accessed in that mode shall reply with ERR acknowledge code to inform the master.

FPI_NO_SPLIT_N

This signal enables or disables split transaction on the FPI-Bus.

- FPI_NO_SPLIT_N = 1: slave is allowed to split (**SPT** acknowledge code)
- FPI_NO_SPLIT_N = 0: slave must not split (no **SPT** acknowledge)

NOTE:
If FPI_NO_SPLIT_N is driven active during bus operation no new split transactions can be initiated but there might be some responses pending.

:FPI_OPC[3:0] (FPI Operation Code)

The opcode bus lines are driven by the granted master in the address cycle to select the type of bus transaction.

Table 3-5: Operation Code Encoding

FPI_OPC[3:0]	Identifier	Description
0000	SDTB	Single Transfer Byte (8 Bit)
0001	SDTH	Single Transfer Half-Word (16 Bit)

Table 3-5: Operation Code Encoding

FPI_OPC[3:0]	Identifier	Description
0010	SDTW	Single Transfer Word (32 Bit)
0011	SDTD	Single Transfer Double-Word (64 Bit)
0100	BTR2	Block Transfer Read (2 transfers)
0101	BTR4	Block Transfer Read (4 transfers)
0110	BTR8	Block Transfer Read (8 transfers)
0111	-	Reserved
1000	SBTR1	Split Block Transfer Request (1 transfer)
1001	SBTR2	Split Block Transfer Request (2 transfers)
1010	SBTR4	Split Block Transfer Request (4 transfers)
1011	SBTR8	Split Block Transfer Request (8 transfers)
1100	SBR	Split Block Response
1101	SBRF	Split Block Response Failure
1110	SBRE	Split Block Response End
1111	NOP	No operation

a. **NOP - No Operation.**
The granted master issues this opcode if no transaction on the bus is requested and no valid address is output. All other opcodes are accompanied by a valid address. There has to be always a master on the bus which is responsible to drive the NOP code in case of an idle bus.

b. **SDTx - Single Data Transfer.**
These opcodes are used to initiate single data read or write transfers. x= B,H,W or D specifies the data width of the transfer.
Single transfer can be split by slave answering with the **SPT** acknowledge code.
If a master gets a **SPT** acknowledge it shall release the bus and wait for one (and only one) split response.

c. **BTRx - Block Transfer Read.**
This opcode initiates a non-split block read transaction of x data items of the maximum data bus width (implementation dependent). x defines the number of items from 2 up to 8. The master will keep the bus and wait for the response. To avoid any interruption it can use a locked transaction.
This transaction might be used for burst reads from fast bus agents, e.g. on-chip page mode memories.

d. **SBTRx - Split Block Transfer Request.**
This opcode initiates a split block read transaction of x data items of the maximum data bus width (implementation dependent). x defines the number of items from 1 up to 8.
Master has to release the bus after the request and wait for the response.
The tranfers might be changed into a series of single accesses if the addressed slave does not support split access.
This transaction might be used for accesses to slow bus agent, e.g. via the external bus controller.

e. **SBR - Split Response.**
If a slave supports split block transfers it drives this opcode during split answers. At the same time it asserts the ID of the former requester on the Tag Bus and the prior sent target address (it's own address) on Address Bus.
A split transfer response can be interrupt by the master itself or by bus arbitration due to a higher priority request.
If the master cannot provide the requested data one after the other it can remain it's request line active and perform idle cycles. This inserts a kind of master wait-states. If another master is granted in between the master with the open response shall continue its response if it is granted again.

f. **SBRF - Split Block Read Failed.**
To inform a former requester that a agent cannot perform a response, e.g. due to a parity error in a bridge, it sends a **SBRF** opcode together with requester tag.
The former requester has to decide to request the transfer again or do some other actions.

g. **SBRE - Split Block Resonse End**
The identifier of the last transfer of such a response is the **SBRE** opcode. It shall be sent together with the last portion of an Split Response.

FPI_RDY (Transfer Termination Indication)

This signal indicates the end of a transfer. It is normally driven active by the selected slave in the (last) data cycle. In this cycle valid information is transferred (data and/or ACK codes). To insert wait-states the selected slave can drive **FPI_RDY** inactive.
(For further information about wait-states refer to 4.2.1"Wait-state insertion," on page 57).

FPI_ACK[1:0] (FPI-Bus Acknowledge Codes)

Acknowledge code bus lines are driven by the selected slave in every data cycle. In case of idle cycles bus control has to drive the acknowledge code.

Table 3-6: FPI-Bus Acknowledge Codes

FPI_ACK[1:0]	Identifier	Description
00	NSC	**No special condition** detected for current data cycle
11	ERR	**Error**, last bus cycle aborted.
01	SPT	**Split**, modify single read transfers to split ones, acknowledges accepted split block requests
10	RTY	**Retry**, slave currently cannot respond. Master has to try later again.

The **FPI_ACK[1:0]** codes have the following meaning (see also Table 3-7):

a. **NSC - No Special Condition.**
This acknowledge code is sent by the selected slave in case of single data transfers and non-split block transactions if no error occurs.
If a slave acknowledges with **NSC** to a **SBTRx** that means that it does not support split transaction. The slave will provide the first data of the requested block together with the **NSC** acknowledge code. The requesting master has transform its block transaction to a series of single reads for the missing data.

b. **SPT - Split.**
A slave acknowledges a split transfer request with **SPT** if it supports split transactions. Otherwise it sends the achnowledge **NSC** to change the request into a series of single transfers.
To modify any single read transaction into a split transaction the slave will answer with the **SPT** acknowledge code when a master sends a single data transfer read. The requesting master has to release the bus and to wait for the answer (with opcode **SBRE**).

c. **RTY - Retry.**
If a slave generally supports the requested transaction but is due to some reason at the moment not capable to deal with it it will acknowledge with **RTY**.
If a master receives a **RTY** acknowledge code from a slave he has to repeat the current transaction. Before doing so, he shall release the bus for at least one cycles (for further details see 4.6“Error Handling,”).

d. **ERR - Error.**
The selected slave terminates the current transaction with error status. (for further details see 4.6“Error Handling,”).

Table 3-7: Acknowledge Codes for certain transactions

Transaction	Acknowledge Code Meaning			
	NSC	SPT	RTY	ERR
Single Data Transfer	no special condition, transfer accepted	convert single transfer to split access, master shall wait for response	busy, try later again	transfer not supported, wrong access
Split Block Transaction	conversion to single data transfers	split accepted, master shall wait for response		
non-split Block Transaction	no special condition, transaction accepted	reserved[a]		

[a] must not be used

3.3 Sideband Signals

The bus specification covers issues like the bus protocol and needed signals

The FPI leaves open opportunity for implementation specific function performance enhancements via "sideband" signals. These signals are loosely defined as any signal not part of the FPI specification. They have meaning only to these agents which are connected to but not the bus itself.

3.3.1 DMA Signals

To enable fly-by DMA functionality dedicated **DMA_SEL_x_N** signals from the DMA controller to DMA destination agents must be implemented. The number of these lines is implementation dependent.

The **DMA_SEL_x_N** signal shall be issued synchronously with the FPI clock.

Table 3-8: Optional DMA Signals

Signal	Setup time		Hold time	
	typ.	worst	typ.	worst
DMA_SEL_x_N	tbd	tbd	tbd	tbd

3.3.2 Interrupt/DMA arbitration signals

For the arbitration of interrupt and DMA requests some lines are routed to any service request node. These signals can be treated as part of the sideband.

The recommended number of lines is in the range of 2 to 4.

NOTE:
An agent that implements a sideband function must exercise great care to avoid interoperability problems with FPI compliant devices that don't comprehend the function.

3.3.3 OCDS Signals

The signal **OCDS_P_SUSPEND** can be used by any On Chip Debug System (OCDS) to stop operation of each peripheral. Source will be a certain module close to the bus. If no OCDS is implemented **OCDS_P_SUSPEND** can be skipped.

This signal shall be routed in parallel to the FPI-Bus.

3.4 FPI-Bus Components

FPI-Bus components are the bus controller, master agents and slave agents.

A FPI-Bus agent needs an interface with master functionality if it is an active system module which performs read and write accesses to other modules. FPI-Bus allows to have more than one bus agent with master interface. In a multi-master configuration an arbitration logic shall be implemented to select one of several competing master interfaces.

3.4.1 Bus Controller (Bus Control)

FPI-Bus needs a bus controller for operation. It can be implemented centralized or decentralized but implementation must ensure the basic functionality:

Bus Arbitration

Due to the multi-master capability the bus arbiter has to select one of the competing masters and grant this one to the bus.

Bus arbitration in a FPI-Bus based system is done by a set of request/grant/lock lines from each potential master to the arbitration logic.

There is one speciality with the FPI-Bus, which is that there has to be always one bus master on the bus, even if no master is requesting the bus. This master is responsible for performing idle cycles which indicates an idle bus.

Again there are several strategies to select the Default Master.

NOTE:
The arbitration scheme itself is not defined by the FPI-Bus specification but is implementation dependent.
An example for an arbiter is given in Appendix 6.4"Bus Arbitration," on page 86.

Idle cycle slave functionality

During idle cycle bus control has act as default slave an provide

- ❑ active **FPI_RDY** and
- ❑ acknowledge code **NSC**.

Time-out Control

The time-out mechanism is intended for bus operations which are not completed by the selected slave with the activation of the **FPI_RDY**. Bus Control shall detect this situation and take the actions which are described in chapter 4.6.1"Time-Out Error," on page 74.

After the (synchronous) release of **FPI_RESET** the Default Master can start an FPI transfer immediately. All other masters have to wait one or two additional cycles because the arbitration round starts again in first cycle.

3.4.2 Who drives the bussed signals

The following table clearifies who drives the bussed signals in certain situations.

Assumptions:

1. Fully decoded address map: In case of an access to a non existing peripheral, the select (fpi_sel_n(0)) is given to a "default slave peripheral".
2. A slave always drives if it gets selected and OPC = $\overline{\text{NOP}}$!
3. Lets assume a design uses block select lines and not all available sfr-addresses are used. Then, the periphal must always drive appropriate FPI signals when it gets selected. Especially in case of waitstates during a read access and in case of an access to a not implemented register.
4. "x" as value specified in the tables below denotes, that the value to drive is arbitrary.
5. The signal fpi_tout is always driven by the bus controler (and therfore not a bussed signal), but it is included here to emphasize, that it is ony activated during timeout.
6. Address cycle and data cycle of the following tranfer will overlap due to the pipelined architecture of the FPI.

Table 3-9: Who drives what

Conditions	Addr Cycle		Data Cycle	
a) during reset a)	fpi_a	0 *by bus control*	fpi_d	0 *by bus control*
	fpi_opc	NOP *by bus control*	fpi_abort_n	inactive ('1') *by bus control*
	fpi_tag	0 *by bus control*	fpi_ack	NSC *by bus control*
	fpi_rd_n	inactive ('1') *by bus control*	fpi_rdy	active ('1') *by bus control*
	fpi_wr_n	inactive ('1') *by bus control*	fpi_tout	inactive ('0') *by bus control*
	fpi_svm	inactive ('0') *by bus control*		
b) 1st cycle after reset	fpi_a	***identical to (e) default grant!***	fpi_d	***identical to (e) default grant!***
	fpi_opc		fpi_abort_n	
	fpi_tag		fpi_ack	
	fpi_rd_n		fpi_rdy	
	fpi_wr_n		fpi_tout	
	fpi_svm			
c) general (no idle)	fpi_a	x *by granted master*	fpi_d	read: x by selected slave write: x by granted master
	fpi_opc		fpi_abort_n	x by granted master
	fpi_tag		fpi_ack	NSC; SPT; RTY; ERR by selected slave
	fpi_rd_n		fpi_rdy	x by accessed slave
	fpi_wr_n		fpi_tout	inactive ('0') *by bus control*
	fpi_svm			

Table 3-9: Who drives what

d) idle cycle	fpi_a	hold previous value or zero by granted master	fpi_d	hold previous value or zero by granted master[b]
	fpi_opc	NOP by granted master	fpi_abort_n	inactive ('1') by granted master
	fpi_tag	master ID by granted master	fpi_ack	NSC by granted master
	fpi_rd_n	inactive ('1') by granted master	fpi_rdy	active ('1') by granted master
	fpi_wr_n	inactive ('1') by granted master	fpi_tout	inactive ('0') *by bus control*
	fpi_svm	inactive ('0') by granted master		
e) grant and no request (default grant)	fpi_a	***identical to (c) general or (d) idle cycle, decision by default master!***	fpi_d	***identical to (c) general or (d) idle cycle according to address cycle***
	fpi_opc		fpi_abort_n	
	fpi_tag		fpi_ack	
	fpi_rd_n		fpi_rdy	
	fpi_wr_n		fpi_tout	
	fpi_svm			
f) wait-state	fpi_a	***identical to (c) general !***	fpi_d	read: hold previous value or x by default slave write: x by granted master
	fpi_opc		fpi_abort_n	***identical to (c) general !***
	fpi_tag		fpi_ack	
	fpi_rd_n		fpi_rdy	
	fpi_wr_n		fpi_tout	
	fpi_svm			

Table 3-9: Who drives what

g) timeout	fpi_a	*identical to (c) general !*	fpi_d	read: hold previous value or x by bus control write: x by granted master
	fpi_opc		fpi_abort_n	x by granted master
	fpi_tag		fpi_ack	NSC by bus control
	fpi_rd_n		fpi_rdy	active ('1') by bus control
	fpi_wr_n		fpi_tout	active ('1') *by bus control* [c]
	fpi_svm			
h) access to non existing peripheral;	fpi_a	*identical to (c) general !*	fpi_d	read: hold previous value or zero *by selected (default) slave* write: x *by granted master*
	fpi_opc		fpi_abort_n	x *by granted master*
	fpi_tag		fpi_ack	ERR *by selected (default) slave*
	fpi_rd_n		fpi_rdy	active ('1') *by selected (default) slave*
	fpi_wr_n		fpi_tout	inactive ('0') *by bus control*
	fpi_svm			

Table 3-9: Who drives what

i) access to non existing register (address) in a peripheral	fpi_a	*identical to* *(c) general !*	fpi_d	read: hold previous value or zero *by selected slave* write: x *by granted master*
	fpi_opc		fpi_abort_n	x *by granted master*
	fpi_tag		fpi_ack	ERR *by selected slave*
	fpi_rd_n		fpi_rdy	active ('1') *by selected slave*
	fpi_wr_n		fpi_tout	inactive ('0') *by bus control*
	fpi_svm			

a. Reset is released with the rising edge of clock. The time from activation of reset to the rising edge after reset has been deactivated is considered case (a). The following clock cycle is considered case (b).

b. The Masters that is owner of the data cycle (was granted in the previous cycle).

c. The owner of the data cycle (granted master of the previous transfer) has to treat a time-out terminated transfer similar to a ERR terminated transfer. At least it shall terminate the transfer.

NOTE:
The difference between (h) and (i) is, the select. In case (h) no existing peripheral is selected. Therefore the default peripheral drives the bussed signals. In case (i) an existing peripheral is selected and it has to drive signals as listed in the tabe.

Debug Support

The Bus Controller has to trace all information given on the FPI bus. In case of error debug information shall be stored within Bus Controller internal registers and an interrupt request shall be asserted. Error in this case means **ERR** acknowledged transactions and **TIMEOUT** terminated transactions.

In order to obtain the data, some FPI signals must sampled at the end of an address cycle while others shall be collected in the corresponding data cycle.

Information stored in the address cycle:

❑ FPI_A[31:0]

❑ FPI_TAG[3:0]

❑ FPI_OPC[3:0]

❑ FPI_RD_N

❑ FPI_WR_N

Information stored in the data cycle:

❑ FPI_D[31:0]

❑ FPI_ACK[1:0]

❑ FPI_RDY

❑ FPI_ABORT

❑ FPI_TOUT

Debug information will be available only for first error. A second error will not rewrite the error registers.

In order to prevent accidental access, read and write of the error registers shall be possible in supervisor mode only (if available). A read of all error registers shall enable a new trace.

3.4.3 Master Agent

A master interface has to provide the following functionality:

Reset

Any activation of the reset signal shall force the master interface into a reset state until the reset signal has been released. Due to the two reset lines four different reset states are possible. These are described in more detail in the Section 3.2.1"System Signals," on page 26.

Idle Mode

All masters that are not granted have to be in idle mode.
An idle master shall keep all its bus drivers deactivated. To get bus ownership a master has to set the request signal (**FPI_REQ_N**) active.

Request/Grant

A master is requesting the bus by activating its request line. To request for consecutive locked bus transfers it has to activate **FPI_LOCK_N** in parallel.
A master is winner of the bus arbitration if its grant line is activated by bus control. It shall start an address cycle an idle cycle in case of grant and **FPI_RDY** active.

Address Cycle

Every bus transfer starts with an address cycle. In this cycle the information about the transfer type and the target slave is given. The master has to drive
- **FPI_OPC[3:0]**,
- **FPI_A[31:0]**,
- **FPI_TAG[3:0]**,
- **FPI_RD_N**,
- **FPI_WR_N** and
- **FPI_SVM.**

A master that wants to perform only one bus access shall release its request/ lock line in the address cycle.

Due to the bus pipeline the address cycle may be executed in parallel to the data cycle of the previous transfer. If this data cycle is not terminated (FPI_RDY inactive) the address cycle must be repeated in the following cycle.

Idle Cycle

Idle cycle is a special kind of address cycle that does not initiate any bus transaction.but indicates an idle bus.
A master which is granted without request or is granted and cannot perform a bus transfer shall perform idle cycles. It has to drive:
- opcode **NOP**
- **FPI_RD_N** inactive,

- **FPI_WR_N** inactive.

Data Cycle

write:
In a write data cycle the master has to
- drive **FPI_ABORT_N**,
- provide data on **DPI_D[31:0]**,
- get acknowledge code from the slave on **FPI_ACK[1:0]** and
- evaluate the **FPI_RDY** signal.

If **FPI_RDY** is not active (wait-state insertion) the data cycle must be repeated in the following cycle.

read:
In a read data cycle the master has to
- drive **FPI_ABORT_N**,
- latch data from **DPI_D[31:0]** (provided by slave),
- get acknowledge code from the slave on **FPI_ACK[1:0]** and
- evaluate the **FPI_RDY** signal.

If **FPI_RDY** is not active (wait-state insertion) the master must neither latch data nor acknowledge code. The data cycle must be repeated in the following cycle.

Due to the bus pipeline the data cycle may be executed in parallel to the address cycle of the following transfer

Time-out

A master (owner of the data cycle) which sees an active time-out signal (**FPI_TOUT**) shall abort the current transaction and release the bus in the following cycle.
For further information refer to Section 4.6"Error Handling,".

Abort

A master which wants to abort an already started transaction can do this by driving **FPI_ABORT_N** active in the (first) data cycle of the transfer. This shall force a slave to cancel the transaction.

3.4.4 Slave Agent

A FPI-Bus agent with slave functionality needs an interface, which is addressed via read and write transfers by master agents.

Each slave interface is selected via a dedicated slave select signal **(FPI_SEL_x_N)**. This select signal is normally decoded from the high order address bits of the address issued by the granted master. An example for the select generation is given in Appendix 6.5"Address Decoder (FPI_SEL_N generation),". A slave is only activated if its **FPI_SEL_x_N** is active and **FPI_RDY** is high.

A slave interface has to provide the following functionality:

Reset Any activation of the reset signal shall force the slave interface into a reset state until the reset signal has been released. Due to the two reset lines four different reset states are possible. These are described in more detail in the Section 3.2.1"System Signals," on page 26.

Idle Mode All slaves that are not selected shall be in idle mode.
An idle slave shall keep all its bus drivers deactivated.

Address Cycle Every bus transfer starts with an address cycle. In this cycle the information about the transfer type and the target slave is given. The slave has to latch the information on:
- **FPI_OPC[3:0],**
- **FPI_A[31:0],**
- **FPI_TAG[3:0],**
- **FPI_RD_N,**
- **FPI_WR_N** and
- **FPI_SVM.**

Idle Cycle:
no actions

Data Cycle The slave controls the data cycle of the transfer. Therefore it has to provide the following information:
- transfer acknowledge code on **FPI_ACK[1:0]** and
- transfer termination information on **FPI_RDY.**

write:
In a write data cycle the slave has to
- transfer acknowledge code on **FPI_ACK[1:0],**
- transfer termination information on **FPI_RDY.**
- latch the data on **DPI_D[31:0],**

If **FPI_RDY** is not active (wait-state insertion) the data cycle must be repeated in the following cycle.

read:
In a read data cycle the slave has to
- latch data from DPI_D[31:0] (provided by slave)
- transfer acknowledge code on **FPI_ACK[1:0]** and
- evaluate the **FPI_RDY** signal.

If **FPI_RDY** is not active (wait-state insertion) the master must neither latch data nor acknowledge code. The data cycle must be repeated in the following cycle.

idle cycle:
The slave has to evaluate the **NOP** opcode and must not perform any action.

Due to the bus pipeline the data cycle may be executed in parallel to the address cycle of the following transfer

Time-out

A selected slave which sees an active time-out signal (**FPI_TOUT**) shall abort the current transaction and release the bus in the following cycle. For further information refer to Section 4.6“Error Handling,”.

Abort

If a slave gets an active **FPI_ABORT_N** in in the first data cycle of a transfer is has to
- abort the current transaction, restore (peripheral) internal registers if possible and
- drive **FPI_RDY** active in the following cycle if wait-states were inserted.

3.4.5 Master-Slave Agents

A bus agent that provides master as well as slave functionality is called master-slave interface.

It is recommended to implement a master-slave interface for every master capable peripheral due to the minimal hardware overhead and the enhanced functionality.

3.5 Default Master

The Default Master will be granted if no bus request is active. It has to perform idle cycles to indicate an idle bus or any other transaction.

There are several schemes possible to select a bus master for this purpose:

- the last bus master might get his GNT line kept active as long there is no other request activated
- one selected implementation dependent master(e.g. CPU) can be the default master, getting the default grant
- the bus controller itself might provide the NOP, being the implicit default master

NOTE:
The default master has the advantage of being able to start a transaction immediately without request. This saves at least one cycle.

3.5.1 Examples of Routing FPI-Bus Signals

Figure 3-1 shows FPI-Bus signal lines routed between FPI-Bus components, a master interface, a slave interface and the central bus control.

NOTE:
It is assumed that the bus arbitration and timout control is done centralized in the bus control which is recommended. The data bus in this example is a 32 bit implementation.

FPI_RESET_N[1:0] and **FPI_CLK** are system signals connected to every bus component.

The bus ownership control lines **FPI_REQ_A_N**, **FPI_LOCK_A_N** and **FPI_GNT_A_N** are point-to-point lines from master A to the bus arbiter (bus control).

The **FPI_TOUT** signal is driven by the timeout control (bus control) and routed to every master and every slave.

FPI_RD_N, FPI_WR_N, FPI_ABORT_N, FPI_OPC[3:0], FPI_A[31:0], FPI_TAG[3:0], FPI_D[31:0], and **FPI_ACK[1:0]** are driven by more than one source (bused signal lines). The figure concentrates on the connection between master A and slave B.

The maximum width of the address bus (normally **FPI_A[31:0]**) depends on the system memory space of the implementation. The upper address bits are mapped to the slave select signals. The lower bits address slave internal memory locations, like registers, FIFOs or RAM.
A master has to be connected to all implemented address lines, a slave only to those address lines which are needed for the slave internal decoding.

The maximum width of the data bus is determined by the largest data type (byte, halfword, word or double-word) which shall be transported over FPI-Bus during normal transfers. The minimum number is determined by the data size of the core (16-bit core => D[15:0], 32-bit core => D[31:0], 64-bit core => D[63:0]).

It is possible to reduce the number of data lines routed to small data size peripherals. However, it has to be insured that the addresses for the peripheral registers then will be located on addresses which are *modulo(*max. data bus width*)*. For example, if an 8-bit peripheral shall be connected only to D[7:0] of a 32-bit bus, its register addresses must be located on word boundaries.

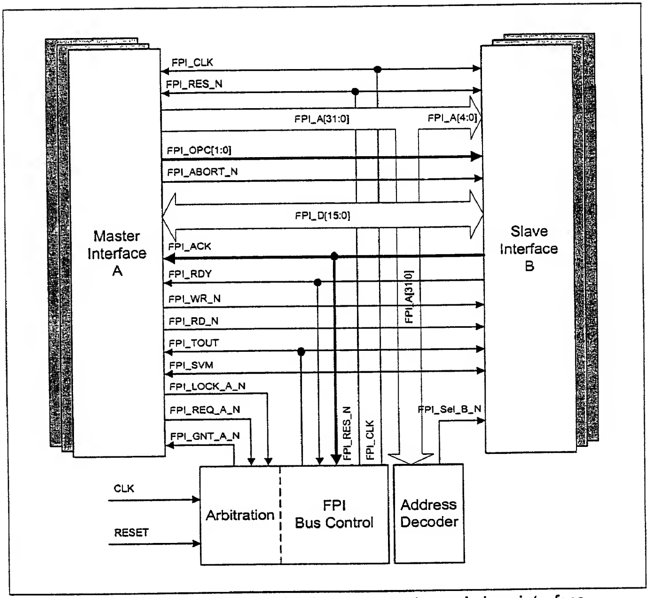


Figure 3-1: Example of routing between master and slave interface (32 bit implementation assumed)

3.6 Power Management

This specification does not cover power management measures. The following proposals may be implemented:

❑ switch off the bus clock whenever the bus is idle

❑ switch off clocking of interfaces that are not active

4.1 Single Data Transfer

The timing diagram in Figure 4-1 shows the basic single data transfers defined on FPI-Bus: read, write and read-modify-write.
Read and write transactions take always 1 cycle (without wait-states) and a read-modify-write access takes 2 cycles.

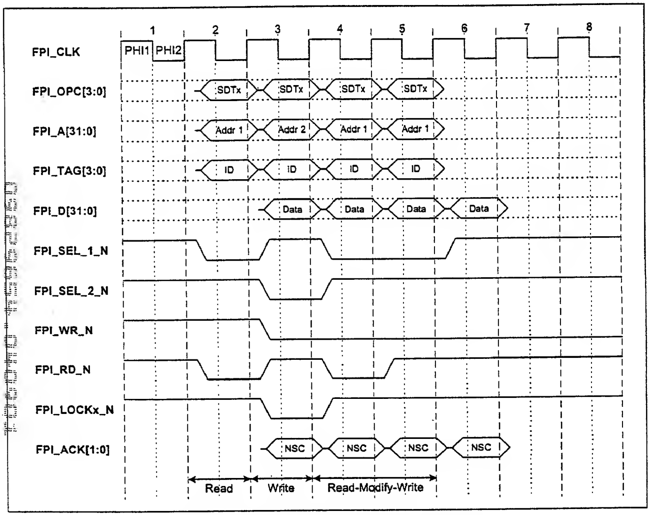


Figure 4-1: Basic Single Data Transaction

Assumptions: **FPI_RDY** is active all the time (no wait-states).

Cycle 2: This is the address cycle of the transfer.
The granted master starts a read data transfers driving the opcode for a single data transfer and setting the read and write control lines to appropriate levels.

The address of the source agent is issued on the address bus and the master ID on the tag bus.
The slave is selected by driving the appropriate select line (**FPI_SEL_1_N**) active in this cycle (by address decoder).

Cycle 3: This is the data cycle of the read transaction started in previous cycle.
The slave selected in cycle 2 drives the requested data onto the data bus, drives **FPI_RDY** active and issues the acknowledge code (**NSC**).

Due to the pipelining the address cycle of the following transaction (a write) is performed in parallel and a new slave is selected by its **FPI_SEL_2_N** line.

Cycle 4: The data cycle of the write is performed.

In parallel the granted master issues the address cycle of the next access (read-modify-write). This access consists of two bus transfers - one read and one write - which have to be locked by the master to avoid any interruption. In the first transfer the data is read from the slave agent. This read is set up in this cycle. Pay attention to the active read and write line. This indicates the read-modify-write. Slaves which cannot deal with this shall treat this access as a normal read.
FPI_SEL_N_1 is activated again.

Cycle 5: The data cycle of the read is performed. The master gets the data from the slave and can start the modification, e.g set bits, clear bits.

In parallel the address cycle of the second transfer (the write-back) is performed.
If the master does not want to perform another transaction it can release its request.

Cycle 6: Due to the released request another master gained control and perform an address or idle cycle (not shown in the figure).

In parallel the data cycle of the write-back is done. The master (the owner of the data cycle) provides the modified data and the slave latches this data at the end of the cycle.

NOTE:
A Read-Modify-Write transaction takes at least two cycles that shall be locked. This is ensured by a locked bus request.
If the master needs additional cycles to modify the data it could perform idle cycles while the bus is still locked.

A Single Data Transfer Read can be split by slave if it supports Split Transactions. Figure 4-2 shows such a scenario.

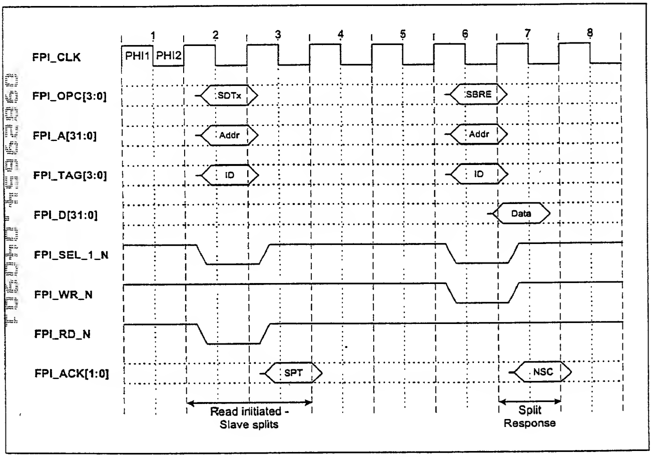


Figure 4-2: Single Data Transfer Split

Cycle 2: A Single Data Transfer Read is initiated in the address cycle.

Cycle 3: The Slaves drives **FPI_RDY** active and answers with **SPT** acknowledge code.
The master has to wait for the split response and it shall release the bus (if it is still granted).
(**FPI_RD_N** is driven high by the default master which does not want to perform any transaction.)

Cycle 4...5: Any other bus action takes place.

Cycle 6: The former slave has gathered the requested data and grabs the bus as master. It performs the address cycle of the split response by driving its own address (to prevent another slave from being selected), the ID of the requester (former master), the **SBRE** opcode and **FPI_WR_N**.

Cycle 7: In this cycle the requested data is transferred.

In parallel the address cycle of the next transaction can be performed.
(**FPI_WR_N** is driven high by the default master which does not want to perform any transaction.)

The sequence on the bus are similar to a Split Block Request except that the decision if split or not is done by slave not by master.

4.2 Single Data Transactions with Wait-states

4.2.1 Wait-state insertion

An active **Ready** signal (**FPI_RDY**) at the end of a cycle indicates that the current transfer is terminated. Valid information will be transferred (data and/or ACK codes).

A slave can insert wait-states if needed by driving **FPI_RDY** inactive during the data cycle. This has the following consequences for the bus protocol:

❑ The master (owner of the current data cycle) has to evaluate this information and
- repeat the data in the following cycle if the current transfer is a write or
- do not latch any (invalid) data if the current transfer is a read.

❑ The granted master (owner of the address cycle) has to repeat the address cycle in the following cycle.

☐ The bus arbiter must not perform a new arbitration round in the current cycle.

Figure 4-3 gives some examples for wait-state insertion.

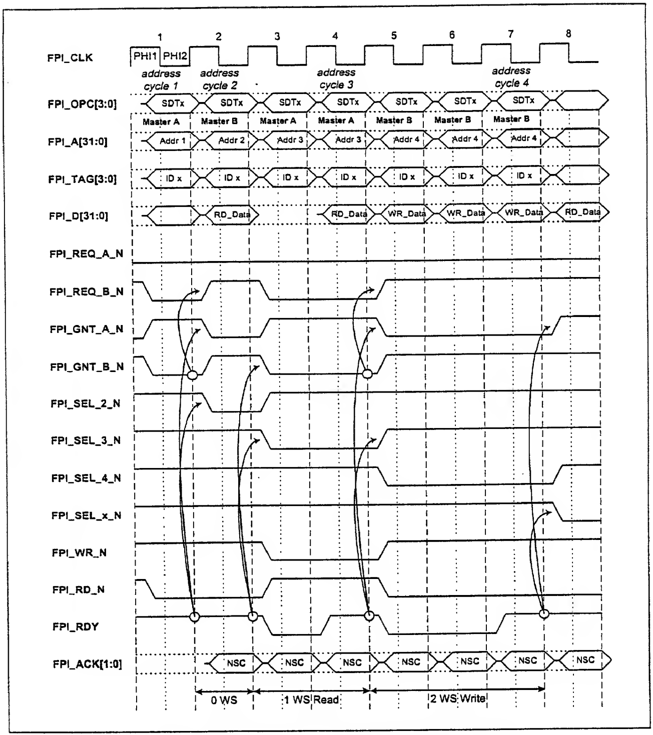


Figure 4-3: Single Data Transfer with Wait-states

4.3 Single Data Transfer Error

If a slave principally cannot execute the required transaction (e.g. wrong data size, access in supervisor mode), it shall respond with an error (**ERR**) acknowledge code.

Figure 4-4 gives an example for this scenario (For further details on error handling refer to 4.6"Error Handling,").

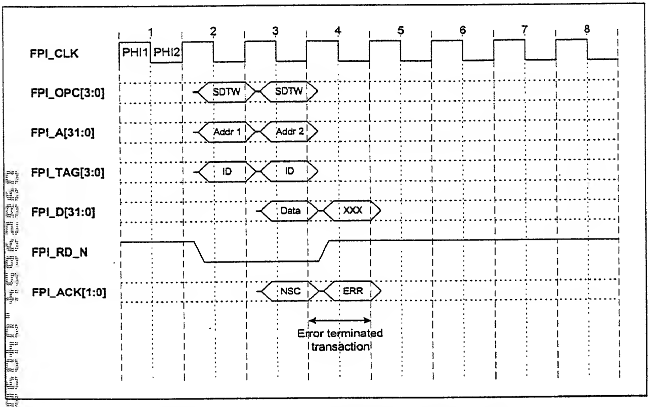


Figure 4-4: Error terminated Single Data Transfer

Cycle 2: The granted master starts a read operation with the address cycle.

Cycle 3: The master (owner of the data cycle) gets the data.

In parallel another read transfer is initiated.

Cycle 4: The master (owner of the data cycle) gets an acknowledge code **ERR** from the slave.
In this example (read transaction) the slave cannot provide valid data. In case of write transactions the data is not latched by slave.

NOTE:
If a slave cannot accept a transaction at the moment, but can generally perform the requested transaction it shall respond with an **RTY** acknowledge code.

4.3.1 Single Data Transfer Retry

If a slave can perform the required transaction generally but not at the moment, e.g. due to a pending split response, it shall respond with the **RTY** acknowledge code.
If a master gets a **RTY** acknowledge code it shall release the bus for at least one cycle.

For further details refer to Section 4.6.4"Retry (RTY) Acknowledge Code," on page 77.

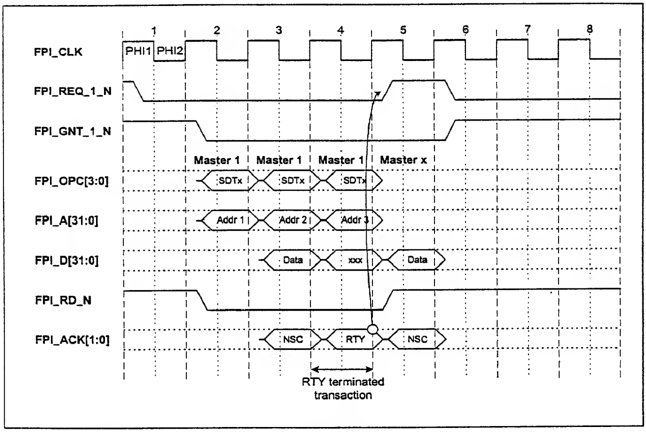


Figure 4-5: Single Data Transfer with RTY Acknowledge Code

Cycle 2: The granted master starts to do some consecutive read Single Data Transfers.

Cycle 4: The master gets an acknowledge code **RTY** from the slave.
The master shall release the bus for at least one cycle.

Cycle 5: The master gets the data from the 3rd transfer (different slave).
The master releases its request and another master may be granted.

Cycle 6: Another master is granted and the master that got the **RTY** issues its request again.

NOTE:
In cycle 5 an out of ourder execution of transfers occurs. To avoid this **FPI_ABORT_N** could be used.

4.3.2 Abortion of started transactions

To ensure the sequence of consecutive bus transfers in case that the slave acknowledges one access with **RTY** the master has the possibility to abort a transaction. All slaves with side-effect registers shall support the abort functionality for read accesses. For write accesses every slave has to support abort functionality.
An example is given in Figure 4-6.

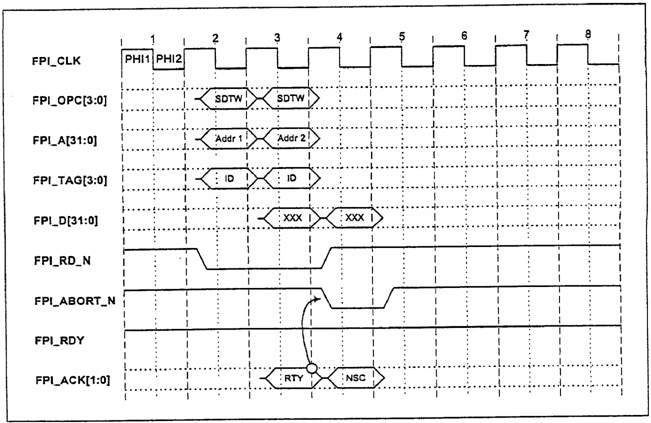


Figure 4-6: Abortion of transactions

Cycle 3: The master gets the acknowledge code **RTY** while the address cycle of the following transfer is already issued.

Cycle 4: The master drives **FPI_ABORT_N** active to enforce the slave to abort the current transfer.
The slave cancles the transfer. In case of read it may provide the requested data but the master will not use it due to the abort. Internally it has to restore the status as if the access did not happen (in case of side-effect registers). In case of write the slave must not take the data.

The master can try to repeat the consecutive accesses later again.

4.4 Block Read Transactions

Two different Block Read Transactions are defined on FPI-Bus:

- ❑ Split Block Read Transactions and
- ❑ Block Read Transactions (non-split).

4.4.1 Split Block Read Transaction

As the name already says, this kind of read transaction is split into two transfers: The response is separated from the request transfer by at least one bus cycle to turn the bus to the slave. Depending on the latency it takes the slave to provide the data this time might be extended. During this time the bus is not blocked and can be used by other agent for transactions.

- ○ During the request information about the target address, the master ID and the number of data items is sent from the master to the selected slave. The request sequence consists of one cycle.
- ○ During the response the former slave gains control of the bus as master and sends the requested number of data items to the former requester identified by its ID. This answer is interruptible if it is not protected by **Lock**.

The maximum number of data items per each block transaction is limited to 8 (refer to Section ": FPI_OPC[3:0] (FPI Operation Code),").

Interruption of Split Block Transfers

The data phase (response) may be interrupted by other transactions. In order to allow the receiving agent properly to accept the next data after an interruption both the initiating master as well as the transmitting slave have to keep track on the data sent, so that they will recognize an interruption, a resumption and the end of the transfer and be able to initiate or receive a new block transfer.

NOTE:
For response end recognition a special opcode **SBRE** is sent during the last data transfer.

Multiple Split Block Transfers.

There may be multiple split block transfers open at a time, depending on the number of split access capable slaves on the bus. Even one slave device may have more than one request open. However, the performance gain is expected to be very small, compared to the additional complexity in this slave.

Therefore it is expected that most split capable slaves will allow only one transaction open at a time (implementation dependent):

If a slave receives another Split Block Read Request while it already has one open, two cases have to be distinguished:

- □ The master ID is the same as the one of the pending response:
    - ○ The target address is different from the previous one:
      In this case the slave will send a RTY acknowledge code, indicating that it cannot accept the request at the moment.
    - ○ The target address is the same:
      In this case the slave has to cancel the pending tranfer and serve the latest request. This is for example necessary to insure a predictable interrupt response times.
- □ The master ID is different from the one of the pending response:
    - ○ In this case the slave will send a RTY acknowledge code, indicating that it cannot accept the request at the moment.

An example for a split block read transaction is shown in Figure 4-7.
It is assumed that the split response is not interrupted.

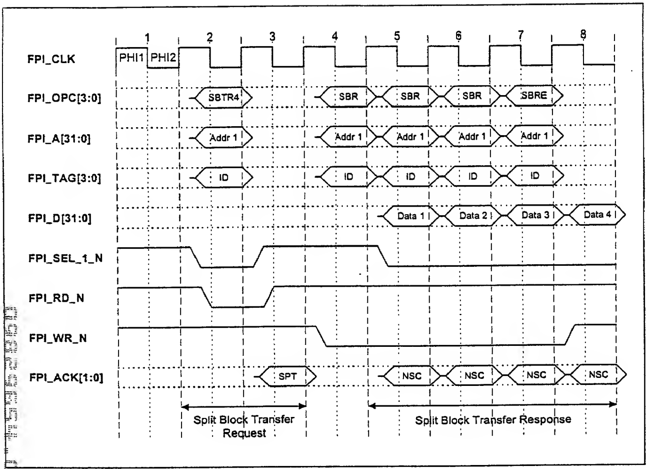


Figure 4-7: Split Block Read Transaction

Cycle 2: The granted master gains control over the bus and issues the Split Block Transfer Request of 4 data items (**SBTR4** opcode) and releases the bus.

Cycle 3: The addressed slave acknowledges with **SPT**. Another other bus operation starts with the address cycle.

Cycle 4: The former slave of the split block transfer request is granted as master. It starts the split response with the address cycle issuing the **SBR** opcode, the former requester ID and its own address. At the same time it drives **FPI_WR_N** active. The transfers are similar to 4 consecutive write transfers but with different opcode and ID. They may be locked.

Cycle 5: The first data issue is sent.

Cycle 6: The second data issue is sent.

Cycle 7: The third data issue is sent together with the **SBRE** opcode. This informs the slave that only one more item will follow. After that the response is terminated.

Cycle 8: The last data item is sent and in parallel a new address cycle of another transfer or an idle cycle is performed.

NOTE:
The data width of the transferred data is always the maximum data bus width (implementation dependent).

4.4.2 Retry of Split Block Read Transactions

A retry of Split Block Read Request only can occur if a bus master is requesting a block from a slave which has already its maximum number of requests (normally 1) open (not finished).
As the example in Figure 4-8 shows the principle is the same as for single cycle retries.

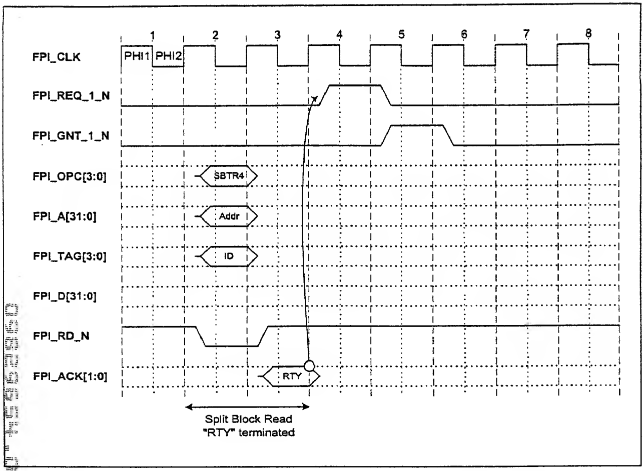


Figure 4-8: Split Block Transfer Request with RTY Acknowledge Code

Cycle 2: A Split Block Read Transaction is initiated by issuing the split transfer request.

Cycle 3: The masters gets the acknowledge code **RTY**. The slave cannot accept the request at the moment and it asks the master to repeat the transfer later again. If the master is still granted it shall release the bus for at least one cycle.

4.4.3 Split Block Transaction Conversion

A slave which cannot perform split block transfers shall convert the split block read to consecutive single accesses. This can be done by answering with **NSC** acknowledge code instead of **SPT**. An example is shown in Figure 4-9.

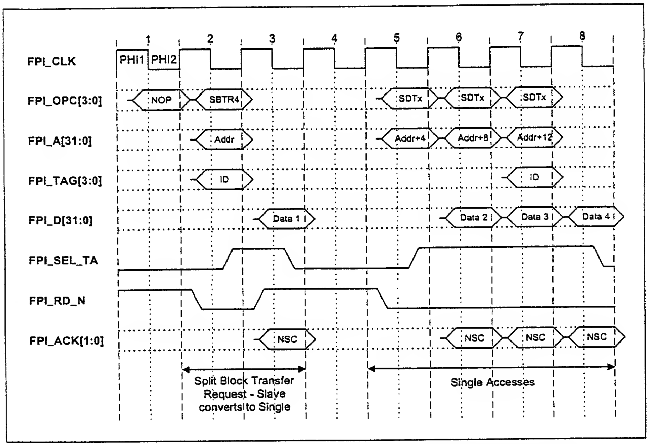


Figure 4-9: Split Block Transaction to Single Access Transformation

Cycle 2:	The Split Block Transfer Request is issued by the granted master.
Cycle 3:	The selected slave drives the **NSC** acknowledge code to inform the master that it does not support split block transactions. (The master shall request the bus again and read the necessary data in consecutive single accesses.) In parallel it provides the first data item.
Cycle 4:	Master requests the bus again.
Cycle 5,6,7:	The missing three data items are fetched with single transfers. Be aware that the address has to be incremented (figure gives example foir 32 bit implementation).
Cycle 8:	Any other transaction can be started.

4.4.4 Non-split Block Read Transaction

This transaction is a kind of consecutive read operations from the same slave which must not be split by the slave.

NOTE:
It is recommended to lock Block Read Transactions to avoid any interruption.

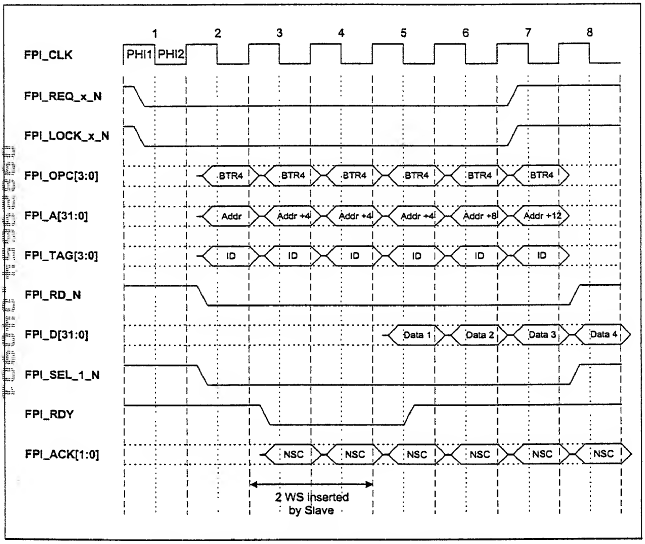


Figure 4-10: Non-split Block Read Transaction

Cycle 2: The granted master initiates a non-split block read transaction of four data items by driving the **BTR4** code and issue the slave address.

Cycle 3, 4: The addressed slave inserts two wait-states because it cannot provide the requested data immediately.

Cycle 5: The transfer of data items starts with the first item.

Cycle 6, 7 The following data items are transferred.
If the slave cannot provide data cycle by cycle it has to insert additional wait-states.

Cycle 8: The last data item is transferred. In parallel a new bus operation starts.

4.4.5 Error during Block Read Transactions

In contrast to an error during single data transfers, the bus master getting an error acknowledge while performing a block transaction must not continue. It has to release the bus.
The bus controller shall issue a CPU trap to propagate the bus error to the system to invoke appropriate measures. Further action are implementation dependent.

Figure 4-11 shows this scenario:

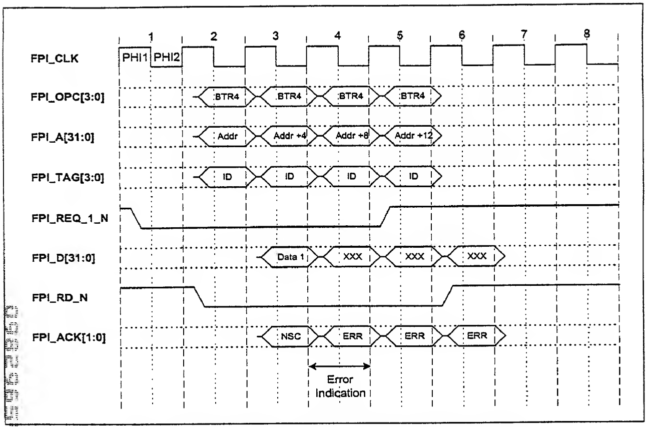


Figure 4-11: Block Transaction with ERR Termination

4.5 DMA Transfers

The purpose of a DMA controller in a system is to do transfers between system units without the overhead of interrupting the CPU with the related store and restore of registers.

Single cycle DMA transfers are possible, when there is a special (hardwired) control line to one of the related agents. In this case the DMA controller will place the address of the source and a read command onto the bus and in addition will tell the destination agent via the special control line to fetch the data from the bus during the same cycle (or vice versa).
This special control line therefore replaces the register address, the read/write control and the select line. It is not part of FPI-Bus but is part of the DMA system implementation.

In a FPI-Bus system there are two possibilities for a DMA control to use the bus for transfers:

- Single cycle transfers between any agent and some special agents (hard-wired solution as described above)
- Normal transfers (2-cycle-DMA), where the DMA controller drives the address, data and read/write lines. In the first cycle the DMA initiates a single read from the source. In the second cycle the DMA asserts the destination address and performs a single write. The DMA has to buffer the data in between these both accesses (see Figure 4-12).

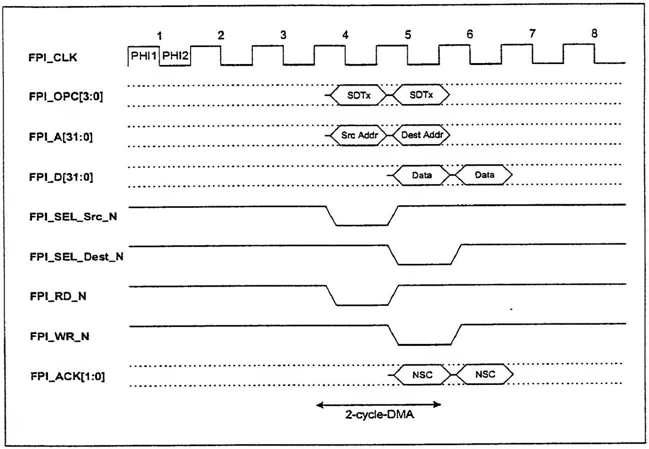


Figure 4-12: Single DMA (in 2 cycles)

NOTE:
Between the read from the source and the write to the destination the data must be buffered inside the DMA Controller.

4.6 Error Handling

Several causes for errors have been discussed. The system has to provide sufficient measures to deal with it and enable error recovery.

Each time an error condition occurred the bus controller will issue a bus error interrupt to the CPU. Debug software then has to resolve the problem or shut down the system safely.

The minimum means to detect and to handle the following errors:

4.6.1 Time-Out Error

Time-out error is detected if too many wait-states have been inserted.
If a slave drove **FPI_RDY** inactive to insert wait-states and does not drive it active again after an implementation dependent number of cycles bus control will issue **FPI_TOUT**.

In this case the selected slave has to abort the current transaction and shall release the bus in the following cycle.

In the following cycle **READY** is driven active again by Bus Control and normal bus operation goes on. For debug purposes some error information should be stored (implementation dependent).

The master which initiated the abnormal terminated transfer has to decide to repeat the transaction or do some other actions due to this abnormal transfer termination.

NOTE:
It should be avoided by implementation that this kind of transaction can be repeated again and again by master otherwise bus might be absolutely blocked and deadlock occurs.

Bus agents which are able to issue Block Read Requests shall internally detect a time-out if the requested transaction is not be answered after a implementation specific time.
For debug purposes some error information should be stored (implementation dependent).

Figure 4-13 shows the time-out handling. The following assumptions (implementation specific) are made:

- time-out detection is done centralized by bus control
- no. of max. wait-states is limited to 3 (if more than 3 wait-states are inserted time-out will be detected)
- the selected slave has started to insert wait-states one cycles ago

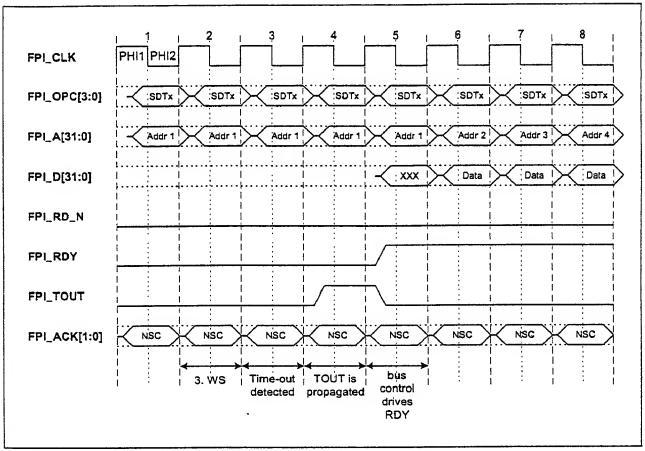


Figure 4-13: Time-out handling

Cycle 3: The 3rd wait-state is inserted still blocking the bus. Bus Control detects the time-out.

Cycle 4: **FPI_TOUT** is driven active by bus control to propagate the time-out.

Cycle 5: The bus control drives **FPI_RDY** active again and releases **FPI_TOUT**.
This enables the bus for operation again.
The master that is waiting for the data must terminate the pending transfer.

Cycle 7: The normal bus operation goes on:
The data cycle of the transaction started before time-out is performed in parallel with the address cycle of the next transaction.

Cycle 8: Normal bus operation.

4.6.2 Access to non existing peripheral / to non existing register(address) in a peripheral

An access to a non existing peripheral or to a non existing register(address) in a peripheral will be **ERR** terminated. This shall cause a system trap and further actions can be taken.

The accessed device either a peripheral or the default slave (bus controller) shall not drive **FPI_RDY** inactive.

Refer to Table 3-9 for detailed information.

4.6.3 Error (ERR) Acknowledge Code

If a master receives an **ERR** acknowledge code it shall abort the current transaction.

The master has to decide to abort the following transaction whose address cycle is already in the pipe by driving **ABORT** active before the end of the address cycle or to perform it. Depending on the transaction it performs the master might get another **ERR**.

If **ERR** acknowledge code is sent a severe error occurs. This could be

- a single transaction of a defined data type which cannot be performed by the selected slave or
- any interface internal error or
- a supervisor mode access to a slave that does not support this mode.

Supervisor Mode Error

As shown in Figure 4-14 a slave that only supports supervisor mode for a given access, and is not accessed in that mode shall reply with **ERR** acknowledge code to inform the master.

The example shows an access to a user mode accessible address in cycle 2/3 and an access to an supervisor mode accessible address in cycle 3/4. The second access is **ERR** terminated because the access does not happen in the appropriate mode.

The **ERR** acknowledge leads to a bus error propagation as described above.

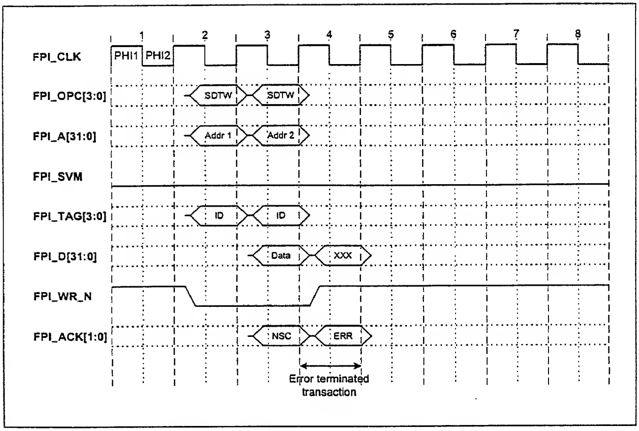


Figure 4-14: Supervisor Mode Access Error

4.6.4 Retry (RTY) Acknowledge Code

The **RTY** code informs a master to repeat the current transaction later again. The master can abort the current transaction (which address cycle has already started) by driving **ABORT** active before end of cycle.

A slave sends a **RTY** if it has already one Split Response open and is gathering data for this response when it receives a new Block Request or Single Transaction.

4.6.5 Split Block Request Failed (SBRF opcode)

If a bus agent cannot respond a Split Block Read request due to some internal errors it performs a special transfer - split block request failed (**SBRF**). This transfer is set up similar to a Split Response with length of one except of that no valid data and the opcode **SBRF** is sent.

A master which receives such a transfer shall not longer wait for a pending split response. Further actions are implementation dependent.

6.1 Deadlock Prevention Rules

6.1.1 FPI Implicit Rules

The following rules are independent of any implementation issues and are derived from the protocol.

1. A master performs only one read or one write access at a time to another device.
   *A master can have only one split response pending at any time.
   (**FPI_TAG[3:0]** bus is the limiting resource)
2. A slave can transform a read access (single or burst) to a split response.
   (not allowed in case of write, read/modify/write or non split burst)
   *Every master must be able to handle a split response.
   (minimal additional hardware effort)
3. Within a combined master/slave device the slave functionality (= another master accesses a register) must never be blocked by the master functionality (= the device itself tries to access another device), i.e. a master/slave device must still be able to accept a normal read/write access or burst request when it waits for a split response.
4. A master which requested and locked the bus must free the bus (= removing request/lock from arbiter) for at least one cycle in case of a "**SPT**", "**RTY**" or "**ERR**" acknowledge code or a time-out condition.
   (in this case the locked access sequence is broken, but a read/modify/write access will still work)

6.1.2 Rider Specific Rules

The following rules describe the implementation of the FPI interfaces on the Rider chip and might vary in the future.

1. A slave, which supports split response, will acknowledge each further access attempt with "RETRY" if it is busy with gathering data for the split response.
   *A slave serves only one master at any time.
2. The EBU has to manage the resource sharing of the external bus which can be owned by one master at any time only. A FPI access to the EBU (respectively the external devices) will be acknowledged with "RETRY", if the EBU is not owner of the external bus at that time.
   *The EBU serves only one master (internal or external) at any time.
3. Loads are always blocking because a slave might transform the read access to a split response (see 6.1.1"FPI Implicit Rules,"). Otherwise the master would have to abort an already started (address phase) second access.

6.2 Rules of Engagement

This chapter is a proposal for Rules of Engagement on the FPI Bus to guarantee Atomic and Lock Operations across different FPI platforms. It is also a compatible super-set of the existing specification

6.2.1 Goals for Atomic Operation

□ Back to back, uninterrupted read followed by a write with no intervening bus operations.

6.2.2 Goals for Lock Operation

□ Deterministic sequence of computation and access of peripherals and data.

□ Ability to ensure completion of preceding bus events.

□ Ensure other masters do not interrupt the sequence of actions in the Lock procedure.

□ Avoid Deadlock

6.2.3 FPI Signals for Atomic and Lock Operation

There are several defined and proposed signals used to build Atomic and Lock Operations.

Table 6-1: Existing Atomic Signals on FPI Bus

Signal	Phase Asserted	Description
FPI_LOCK_x_N	Arbitration Round (concurrent with FPI_REQ_x_N)	A Locked Bus cycle is requested from the Arbiter. No other Masters should be granted Bus control until this Master de-asserts FPI_REQ_x_N
<FPI_RD_N, FPI_WR_N>	Address Cycle	Both signals asserted = 00 during a valid Address Cycle signify a Read-Modify-Write operation

Table 6-2: Proposed Lock Signals on FPI Bus

Signal	Source	Phase Asserted	Description
FPI_NO_SPLIT_N	Arbiter	Address phase (or more)	A Slave device may not acknowledge a read with a SPT (Split Response).
FPI_RES_REQ_x_N	Slave	Arbitration Round (in place of FPI_REQ_x_N)	Request signal for an FPI bus cycle for a Split Response. Only asserted by a Master/Slave that is able to perform a Split Response
FPI_BUSY_x_N	Slave	--	Slave capable of Split Response is currently busy completing a Split Response. Signal is de-asserted when the operation is complete and the device has returned the Split Response.

6.2.4 FPI_NO_SPLIT_N:

□ The arbiter asserts FPI_NO_SPLIT_N when the bus will be, or is granted to a master that has asserted FPI_LOCK_x_N during the arbitration phase. It will remain asserted until the granted master has de-asserted the FPI_LOCK_x_N signal, even if the FPI_REQ_x_N has been periodically de-asserted.

□ Slaves may not respond with a SPT response (Split) while FPI_NO_SPLIT_N is asserted, but they may complete already pending Split transactions.

□ A system desiring slaves to never assert SPT may permanently tie this line asserted.

6.2.5 FPI_RES_REQ_x_N:

□ Implemented *only* by Master/Slave interfaces capable of responding with SPT. (Split Response)

□ Each Master / Slave capable of a Split Response uses FPI_RES_REQ_x_N to signal a bus request to the Arbiter for a Split Response bus cycle. It only uses the FPI_REQ_x_N signal to request the bus for a Master initiated bus transaction, not a Split Response.

6.2.6 FPI_BUSY_x_N:

□ Implemented *only* by Master/Slave interfaces capable of responding with SPT. (Split Response)

□ Each Master / Slave capable of a split response uses FPI_BUSY_x_N to signal it is busy completing a Split Response. The signal is asserted as soon as the Slave recognizes it is performing a Split Response, and is de-asserted when the pending response has completed successfully on the FPI bus.

6.2.7 FPI Arbiter Rules:

- ❑ The Arbiter may not change ownership of the bus grant to another Master once given if the FPI_LOCK_x_N was asserted during the request cycle. This is a "sticky" ownership. It may only grant to another master when the FPI_LOCK_x_N signal is de-asserted.
- ❑ The arbiter asserts FPI_NO_SPLIT_N when the bus has been granted to a master that has asserted FPI_LOCK_x_N during the arbitration phase. It will remain asserted until the granted master has de-asserted the FPI_LOCK_x_N signal, even if the FPI_REQ_x_N has been periodically de-asserted.
- ❑ When the Arbiter receives a FPI_LOCK_x_N, and that Master has won arbitration, the Arbiter asserts FPI_NO_SPLIT_N. It does not yet assert FPI_GNT_x_N to that master. It may not grant the bus to other masters. It allows any FPI_RES_REQ_x_N (Split Response) to be granted while it waits for all FPI_BUSY_x_N signals to become de-asserted. At that point it is allowed to give the FPI_GNT_x_N to the Locking master.
- ❑ The EBU may have special cases.

NOTE:
In implementations without any split transaction support the arbiter does not need to generate the FPI_NO_SPLIT_N signal.

6.3 Data Alignment

Agents have to transfer bytes and halfwords on their natural byte and halfword lanes (see **Figure 2-2 on page 22**). The byte and halfword ordering scheme is little endian.

Accesses to agents that do not support the full data bus width shall be split by master into consecutive accesses of the supported width. The bus protocol will not cover that feature.

NOTE:
In the SFR address range (peripheral accesses) word accesses on halfword boundaries are not allowed.

6.4 Bus Arbitration

Bus arbitration in a FPI-Bus based system is done by a set of request, lock and grant lines from each potential master to the arbitration logic.

6.4.1 Example for a Bus Arbiter

For arbitration each potential master(-interface) on FPI-Bus has one request and one grant line connected to bus arbiter. Masters that want to lock the bus for a certain number of consecutive transfers need an additional lock line (**FPI_LOCK_x_N**).

The request lines have different priorities that can be fixed or programmable by a set of registers inside the arbiter.

Arbitration Control does an arbitration round if any request is active and the bus is not locked and **FPI_RDY** is active.
At the end the **FPI_GNT_N** signal of the master with the highest priority of all requesting masters will be driven active.

Definition: A master is granted in cycle N if its dedicated **FPI_GNT_N** line and **FPI_RDY** was active in cycle N-1. It owns the data cycle starting in cycle N+1 ending with the cycle in which the transaction is terminated either by activating **FPI_RDY.**

If there is no request the **Default Master** will be granted.

There are several schemes possible to select a bus master for this purpose:

- the last bus master might get his grant line kept active as long there is no other request activated

- one selected implementation dependent master(e.g. CPU) can be the default master, getting the default grant
- the bus controller itself might be the implicit default master

NOTE:
Important is that there has to be a master on the bus for every cycle.

Figure 6-1 gives an example of an arbitration scenario.
It is assumed that the priority is decreasing from master A to master C. The default master has lowest priority.

Cycle 1: Default Master owns the bus, master A, B and C are requesting for the bus.

Cycle 2: Master A gains control over the bus because it has the highest priority and issues one opcode. Master A releases its request while B and C are still requesting.

Cycle 3: Master B gains control over the bus. Its priority is higher than the one of C. B releases its request and issues one opcode. Master C is still requesting for the bus.

Cycle 4: Master C gains control over the bus. It issues one opcode and still requests the bus to do some more transactions. At the same time master A asserts a new request.

Cycle 5: Due to the mentioned priority master A gets the bus again. It issues one opcode and releases its request. Master C is still requesting for the bus.

Cycle 6: Master C gains control over the bus again and issues another opcode. It is still requesting for the bus while there is no other request.

Cycle 7: Master C still owns the bus and issues one more opcode. It is still requesting for the bus while master B asserts a new request.

Cycle 8: Master B gains control over the bus again and issues one opcode. It is still requesting for the bus while C does the same. Because there is no request from A master B will own the bus in the following cycle, too.

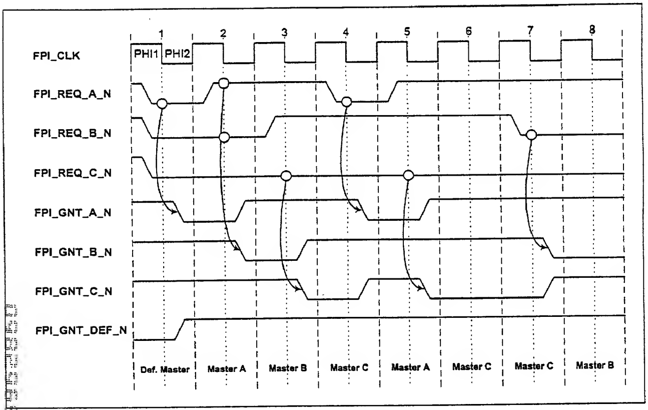


Figure 6-1: General Arbitration Scheme

NOTE:
To enable single cycle master switch bus arbiter must evaluate all request signals with the falling clock edge or combinatorial and drive a grant line before end of the cycle.
If the request signals are evaluated at the leading edge of the following cycle the master switch takes two cycles. That means the former master has to perform an idle cycle before a switch.

Implications of FPI_LOCK_N and FPI_RDY

❑ FPI_RDY

The arbiter shall not take care of the **RDY** signal. This has to be done by the master itself.

A granted master shall evaluate the **RDY** signal and must not start any bus actions until **RDY** is active. If **Ready** is active the granted master can start bus transactions in the following cycle.

❑ FPI_LOCK_N

A activation of **FPI_LOCK_N** will prevent a new master to gain control over the bus, even if it has a higher priority than the current bus master. The following Figure 6-2 shows the related protocol details.

Assumptions for this example:
There are three masters, a Default Master and two other masters, called A and B.
The priority of the masters is fixed: master A has the highest priority then follows B and the Default Master at the lowest.
The figure shows the case that a master activates the **FPI_LOCK_N** to prevent interruption by a higher priority master.

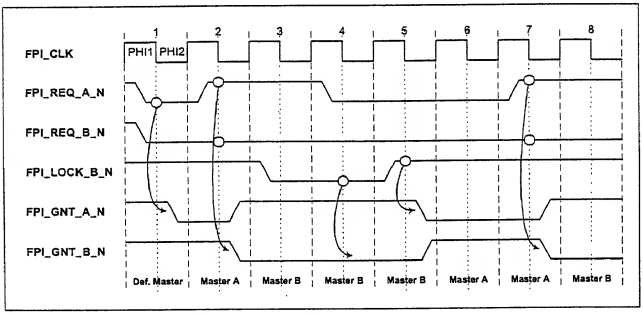


Figure 6-2: Arbitration in Case of Lock

Cycle 1: Master A and B are requesting for the bus.

Cycle 2: Master A gains control over the bus because it has a higher priority and starts one transaction. Master A releases its request while B is still requesting.

Cycle 3: Master B gains control over the bus. It starts another transaction and is still requesting for the bus. In parallel it drives its **Lock** line active.

Cycle 4: Master B is still granted. Due to the **Lock** the bus arbitration is stopped until **FPI_LOCK_N** is released again.

It still requests for the bus while at the same time master A asserts a new request.

Cycle 5: Although master A was requesting with higher priority master B is still granted due to the stopped arbitration.
Master B releases its **Lock** line but keeps its request active. Normal bus arbitration goes on.

Cycle 6: Master A gains control due to its higher priority.

Cycle 7: Master A still has control over the bus. It initiates another transaction and releases it request.

Cycle 8: Master B is granted because its still active request and the released request of master A.

6.5 Address Decoder (FPI_SEL_N generation)

Each slave interface has one dedicated **FPI_SEL_N** input.

Although the address decoding is not part of this specification it is an important issue due to the fact that only a selected slave is active on the bus.

The implemented address decoding scheme has to ensure:

- that at the end of an address cycle the appropriate **FPI_SEL_x_N** is active,
- that only one **FPI_SEL_x_N** line is active at the same time.

For security reasons it is strongly recommended to fully decode the implemented memory map. Figure 6-3 shows an example.

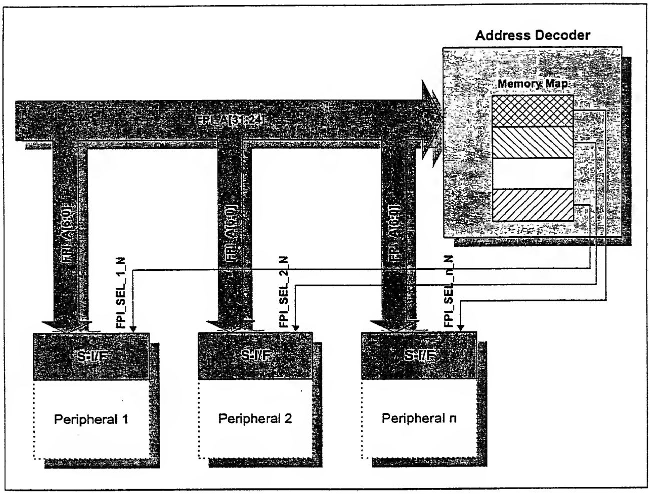


Figure 6-3: Example of Select Generation

6.6 Coupling of two FPI-Bus based Devices

To connect two FPI-Bus based devices the External Bus Controller (EBU) may be used. This unit puts the FPI access ontó the external bus. The EBU of the receiving device uses its EBU to sample the access on the external bus and propagates it to the internal FPI.
The handshake of both EBUs may be done with dedicated EBU signals like chip-select, byte-enable etc.

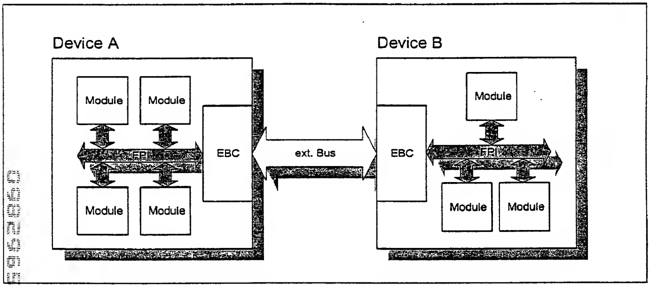


Figure 6-4: Coupling of two FPI based devices via EBU

Appendix D

Title:
BPI Specification
Draft Version 0.9

1. General

This document describes how to use the Bus-Peripheral-Interface. In the current version, this document is about BPI for FPI bus only.

2. Function

2.1. General concept

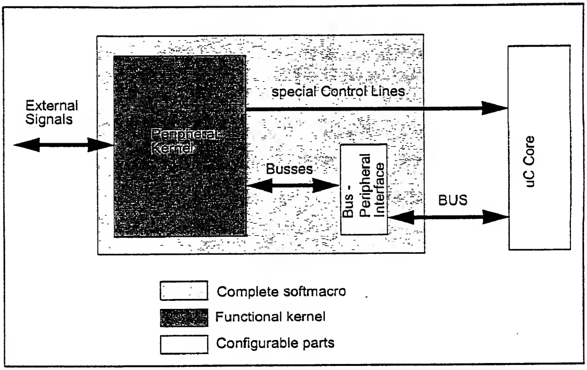
A Bus-Peripheral Interface BPI completely handles the bus protocol of the chip internal interconnect bus; this BPI can therefore be seen as the standardized interface to the FPI-Bus.

On the other side all BPIs adhere to the fundamentals given in "Platform Concept: SMIF Specification" (T. Steinecke, P. Schneider) and considered binding for all platform peripheral.

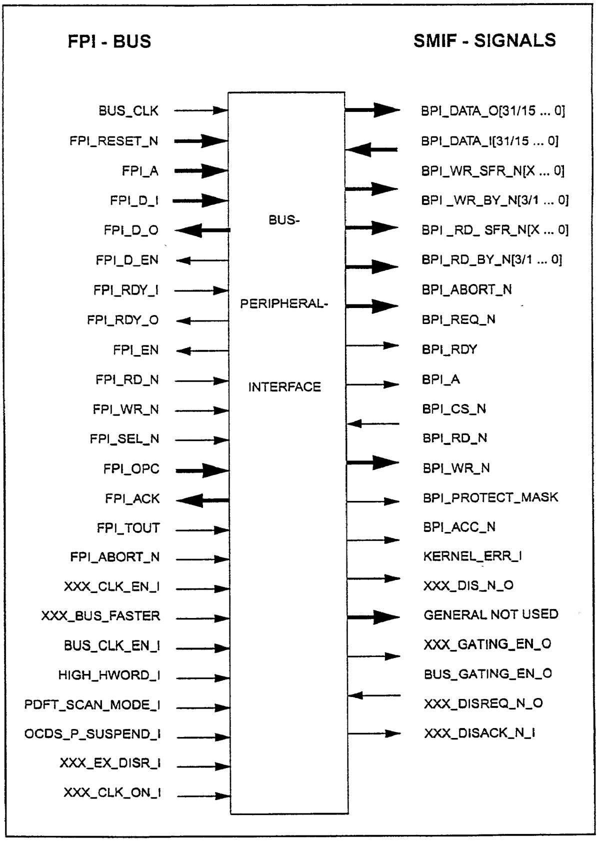
The peripheral's functional code (named "kernel" throughout this document) is thus independent of the used bus protocol.

The main tasks of the BPI are

- Buffering — To define a predictable bus timing some standardisation on fan out and fan in is mandatory
- Decoding — As Kernel SFR's will have different addresses in different uC's, addresses are decoded inside BPI mainly.
- Handshaking — Bus and peripheral may use different clock rates (even dynamically !). Thus some way of synchronization has to be provided.
- Error Handling — Access to undefined locations and unclocked peripherals are functions not regarded legal under normal circumstances; some error notification via the bus may be needed.
- Test Mode — If special test modes are necessary the should be implemented likewise throughout a chip.


External Signals
Peripheral Kernel
special Control Lines
Busses
Bus - Peripheral Interface
BUS
uC Core
Complete softmacro
Functional kernel
Configurable parts

2.2. I/O signals from BPI_INTERFACE



The FPI signals are described in the in the FPI-Specification.

Description of the interface ports to the SMIF:

BUS_CLK	Corresponds to the fpi_clk
fpi_rdy_i	Corresponds to the fpi_rdy signal (fpi_rdy_b) from the FPI
fpi_rdy_o	Ready acknowlege from the BPI. Driven to the signal fpi_rdy_b if enabled with the fpi_en_o signal.
fpi_en_o	Enable signal for the fpi_rdy_o and fpi_ack_o tristate driver
fpi_ack_o	Acknowlege signal from the BPI. Driven to the signal fpi_ack(_b) if enabled with the fpi_en_o signal. **Note: Must be INOUT for synopsys test compiler**
fpi_d_i	Corresponds to the fpi_d from the FPI
fpi_d_o	Driven by the BPI internal data register. Driven to fpi_d(_b) if enabled with the fpi_d_en_o signal.
fpi_d_en_o	Enable signal for the fpi_d_o tristate driver
BPI_DATA_I[31/15...0]	Input data bus, either 32 or 16 bits wide. Source: Peripheral kernel port kernel_data_o
BPI_DATA_O[31/15...0]	Output data bus, either 32 or 16 bits wide. Destination: Peripheral kernel port kernel_data_i
BPI _RD_SFR(x...0)	Dedicated read signals, one each for each SFR.
BPI_RD_BY_N[3...0] (optional)	Selection which bytes have to be read
BPI _WR_ SFR(x...0)	Dedicated write signals, one each for each SFR.
BPI_WR_BY_N[3...0]	Selection which bytes have to be written
BPI_PROTECT_MASK [31/15...0] (optional)	Mask for bit-protection. ‚0‘ means that the BIT must not be changed ‚1‘ means that the BIT must be changed in the peripheral.
BPI_ABORT_N (optional)	This signal is directly mapped to the FPI-BUS-signal FPI_ABORT_N. Each peripheral can abort any access in this way. **In the current version this signal is not used by the interface!! This is wrong and will be resolved in the next version**
BPI_REQ_N (optional)	Indicates a valid access if dynamic waitstate insertion is required (handshake_c=1). This signal is reset by BPI_RDY.

BPI_RDY (optional)	Ready indication from the kernel after a dynamic waitstate insertion. While this signal is driven ‚0', it is not possible to request a new access with the signal BPI_REQ_N. Exactly one waitstate is inserted if BPI_RDY is directly bridged to BPI_REQ_N in the kernel
BPI_A (optional)	Synchronized address bits (latched FPI_A) for the RAM-Interface. Only valid if a RAM-Interface is configured
BPI_CS_N (optional)	Chip select signal for the RAM-Interface
BPI_RD_N (optional)	Read signal for the RAM-Interface
BPI_WR_N (optional)	Write signal for the RAM-Interface
BPI_ACC_N	This signal indicates to the kernel when a Write or Read access will take place. It is active for the period of one "Master clock" before and during the rising edge of the peripheral clock. With this signal, multiple reads or writes to a peripheral register can be prevented. A normal read cycle doesn't need this signal, but if a register performes destructive read accesses this signal must be considered.
XXX_BUS_FASTER	Notifies the BPI module of a peripheral that the bus clock rate is currently higher than the peripheral clock rate. This speed indication is needed to perform zero Waitstate write accesses to the peripheral's internal registers whenever the Bus clock rate is equal to or slower than the peripheral clock rate. Connecting XXX_BUS_FASTER to static '1' will insert exactly one waitstate into each access.
KERNEL_ERR	Error signal from the Peripheral Core to the BPI module signalling an error condition during a not allowed RD/WR access from BPI module to the peripheral core. Must be connected to ‚0' if not used. **In the current version this signal needs to be driven before an access will be performed. This is not possible if the peripheral clock is turned off !**
OCDS_P_SUSPEND	Notifies the peripheral to stop the peripheral clock for debugging purposes.
xxx_ex_disr	Requests the peripheral to stop the peripheral clock.
Bus_clk_en	Enable signal for the BPI component clock domain. Is used for clock gating.
xxx_clk_en	Enable signal for the peripheral kernel clock domain. Is used for clock gating.
Bus_gating_en	Enable signal for clock gating of the peripheral clock domain.

xxx_gating_en	Enable signal for clock gating of the peripheral clock domain.
xxx_clk_on	Special function pin for RTC. Must be connected to ,1' if not used!
xxx_dis_n_o	Special function pin for RTC. Leave open if not used.
xxx_disreq_n	Disable request signal to the kernel to switch off the peripheral clock.
xxx_disack_n	Disable acknowledge signal from the peripheral kernel to allow switching off of the peripheral clock. Bridge this signal to xxx_disreq_n if no specific function is implemented in the kernel
high_hword	**This signal is only considered during write accesses if the fpi data bus width is 32 bits and the kernel data bus width is 16 bits.** High_hword = „0" means that the lower 16 bits of the fpi bus carry valid data and are to be mapped to the kernel data bus. High_hword = „1" means that the upper 16 bits of the fpi bus carry valid data and are to be mapped to the kernel data bus.

2.3. Architecture of the whole peripheral and testbench

A peripheral is a hierarchical construction (see Figure 2 below).
The peripheral kernel can be given either as a structural or as a rtl description.
The components <project>_bpi and <project>_kernel are contained within the module <project>_syn. (syn means synthesis unit)

Example: p3_wdt_bpi and p3_wdt_kernel comprise p3_wdt_syn.

On this project level, no tristate drivers are included, they are instantiated in module <project>_bus_driver.

Example: p3_wdt_bus_driver.

For the distribution of the master clock to the BPI and its kernel a special clock driver each will be instantiatied inside module <project>_clock_gating.

Example: p3_wdt_clock_gating.

The three modules <project>_bus_driver, <project>_clock_gating and <project> _syn are contained inside <project>

The dedicated testbench (<project>_tb) for this toplevel module consist of the peripheral itself (Example: p3_wdt)and the test units FPI (bus model) and <project>_IOC_top (input output control).

Example: p3_wdt_ioc_top.

The figure below shows the internal hierarchy of the bpi_interface and the other components.

Remark: All VHDL units should have a unique prefix in their name to signal their belonging to a certain peripheral. *p3_xxx* is used throughout this document as generic peripheral name.

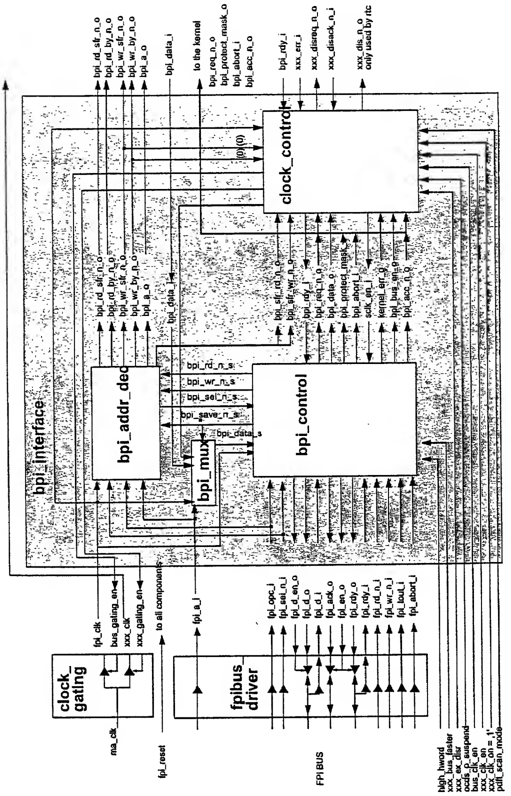


Figure 1: BPI architecture

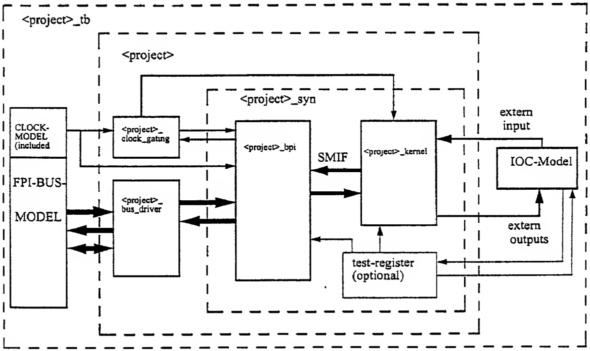


Figure 2: Architecture of the whole peripheral and testbench

If you use this example proceed as follows:

- Complete the component <project> and<project>_syn with the external input and output signals given by your<project>_kernel.
- Extend the signal list and modify the port map in file <project>_tbe_tba-a.vhd.
- Change the constant nr_of_pins_c and edit the constant pin map of the <project>_IOC_top port map

The <project>_tbe_tba-a.vhd contains an example.

3. Parameter setting in the project package

The configuration of the BPI is handled in the BPI_PACKAGE, contained in the file <project>_bpi_pack-p.vhd.
For every peripheral this package needs to be copied into the project packages directory under a unique name, usually by prefixing the original name with the project prefix, e.g. *P3_XXX*_BPI_PACK-P.VHD. Configuration of the BPI is then accomplished by editing (extending) this file. One important point is editing the name of the package likewise !

> Example for the watchdogtimer (WDT):
> $HWPROJECT/VHDL_PACKS/RTL/P3_WDT_BPI_PACK-P.VHD.

The personalised project package is then imported by all other units with the statement

USE WORK.*P3_XXX*_BPI_PACK.ALL;

Inside this package several constants are to be checked and changed if necessary:

fpi_addressbuswidth_c	Indicates the width of the incoming FPI_A - BUS Preset to 32 Bit. Normally no change necessary
fpi_databuswidth_c	Indicates the width of the bidirectional FPI_D - BUS Preset to 32 Bit. Normally no change necessary
kernel_databuswidth_c	Indicates how many bits are provided by the largest peripheral register in one access. Possible values are 32, 16, 8. The constant sfr_size_c indicates the numbers of bytes and is calculated from kernel_databuswidth_c by: sfr_size_c := kernel_databuswidth_c / 8
addressbuswidth_c	The constant addressbuswidth_c indicates how many bits are to be decoded by the address decoder for the SFR select signals. In order to get the correct address of the register the bits fpi_a(fpi_high_c downto fpi_low_c) will be decoded. Note that fpi_a(1 downto 0) are reserved for the byte selection. The default value is 8; {(7 downto 0)}
fpi_low_c	LSB of the decoded address for the SFR addressrange. Preset to 2. Normally no change necessary
fpi_high_c	MSB of the decoded address for the SFR addressrange. Preset to 7. Normally no change necessary
handshake_c	When you need a handshake-procedure between interface and kernel, set the constant handshake_c:=1; Preset to 0
fpi_addr_width_c	Range for bit_vector. Preset to 8. No change necessary unless the constant addressbuswidth_c was changed. This constant is only used to define the type of the following constant sfr_addr_c.
sfr_addr_c	Array of SFR-Addresses for the address decoder. Note that the first 4 addresses (x'00,x'04,x'08,x'0C) are predefined !
destructive_read_c	Indicates whether the peripheral includes destructive read registers. Preset to 0

dest_addr_c	Subarray of SFR-Addresses (must also be present in sfr_addr_c) which are destructive read registers. Preset to „10“. This constant is only relevant if destructive_read_c = 1. **If only one register is dest.read insert it twice**

The next group of constants is relevant if a RAM-interface is required only.

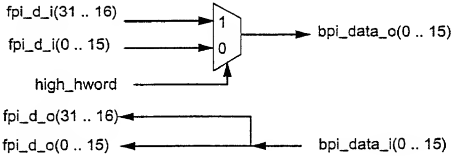
ram_interface_c	Gobal indication if a RAM is required! Preset to FALSE
ram_addr_c	The constant indicates the bit combination which shall create the select signal for the ram inside the kernel. Preset to „1“ This constant is compared to fpi_a(ram_high_c downto ram_low_c) **If ram_high_c is not equal to ram_low_c define a subtype !**
ram_addr_low_c	Indicates the lsb of the RAM address. Preset to 0
ram_addr_high_c	Indicates the msb of the RAM address. Preset to 7 bpi_a<=fpi_a(ram_addr_high_c downto ram_addr_low_c) Preset to bpi_a<=fpi_a(7 downto 0)
ram_low_c	LSB of the decoded ram address. Preset to 8. Normally no change necessary
ram_high_c	Preset to 8 ; -- only 256 byte RAM If you set this constant, for example to the value 9 then don't forget to change the constant ram_addr_c to a 2-Bit-Vector. Example: std_ulogic_vector(ram_high_c downto ram_low_c) := „10“ This sector indicates if a RAM or SFR are selected.
ram_databuswidth_c	Preset to 32 -- the same as the FPI data bus. You can change it to 16 or 8 for another ram databuswidth. **!! IN THE CURRENT VERSION !!!!** **!! ram_databuswidth_c must equal kernel_databuswidth_c!** **!! It is no own data input for the RAM inserted !!**
ram_access_c	The result of the calculation ram_databuswidth_c / 8

If a RAM-module is not configured (ram_interface_c = FALSE), the RAM special signals must not be used in the kernel. These are

BPI_A_I	the registered fpi_a(ram_low_c-1 downto 0)
BPI_WR_N_I	write signal for the RAM
BPI_RD_N_I	read signal for the RAM
BPI_CS_N_I	chipselect signal for the RAM

4. Binary and Enhanced Mode

For compatibility reasons some bus mapping is provided. This is controlled by the special signal high_hword. Only if the FPI databus width is 32 and the Kernel databuswidth is 16 is it possible to multiplex the upper or the lower16 Bit of the FPI databus to the kernel databus independently from the lsbs of fpi_a ! Nomally this signal must be held '0'.



5. The FSM of the controlunit

5.1. control_m1

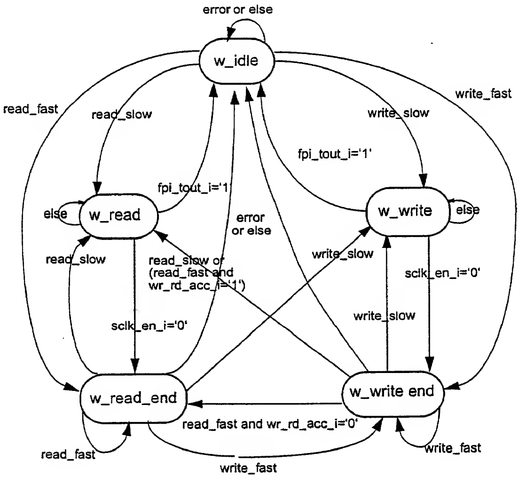
read_fast <= xxx_bus_faster_i='0' and fpi_rd_n_i='0' and fpi_opc_i /= „1111" and
fpi_sel_n_i='0' and fpi_rdy_i='1' bpi_sel_n_i='0';

write_fast <= xxx_bus_faster_i='0' and fpi_rd_n_i='1' and fpi_opc_i /= „1111" and
fpi_sel_n_i='0' and fpi_rdy_i='1' bpi_sel_n_i='0' and fpi_wr_n_i='0';

read_slow <= (xxx_bus_faster_i='0' and fpi_rd_n_i='0' and fpi_opc_i /= „1111" and
fpi_sel_n_i='0' and fpi_rdy_i='1' bpi_sel_n_i='0')
or (read_fast and bpi_dest_i='1';

write_slow <= xxx_bus_faster_i='0' and fpi_rd_n_i='1' and fpi_opc_i /= „1111" and
fpi_sel_n_i='0' and fpi_rdy_i='1' bpi_sel_n_i='0' and fpi_wr_n_i='0';

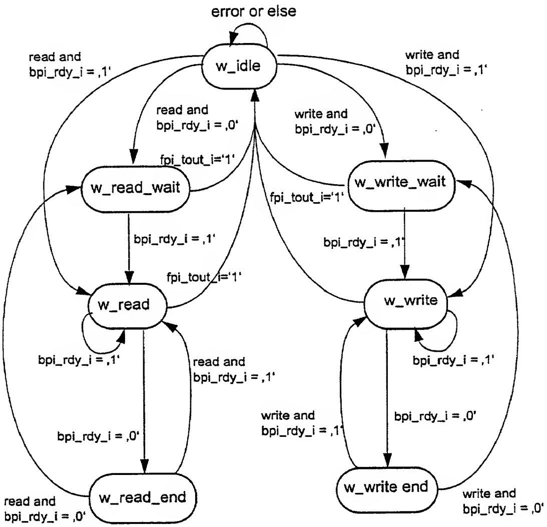
error <= fpi_tout_i='1' or kernel_err_i='1';



5.2. control_h (handshake)

read <= fpi_rd_n_i=‘0‘ and fpi_opc_i /= „1111“ and
fpi_sel_n_i=‘0‘ and fpi_rdy_i=‘1‘ bpi_sel_n_i=‘0‘

write <= fpi_rd_n_i=‘1‘ and fpi_wr_n_i=‘0‘ and fpi_opc_i /= „1111“ and
fpi_sel_n_i=‘0‘ and fpi_rdy_i=‘1‘ bpi_sel_n_i=‘0‘



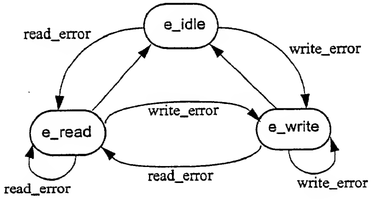
5.3. FSM for error handling

An error occured if

- the peripheral is selected but the SFR address is wrong,
- a access is performed while the kernel clock is switched off or
- a access is performed while the kernel indicates a kernel error

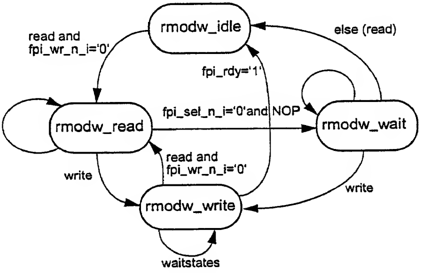
In the e_read state the fpi data bus nonetheless must driven by the peripheral.
If the kernel clock is switched off only accesses to the clock control register is allowed.

In case of timing problems there is a constant error_handling_c **defined inside the architecture p3_xxx_bpi-rtl-a.vhd to switch off this errorhandling** (constant error_handling_c: boolean:=false;)**!** **In this case the fpi_ack is in everytime NSC!**

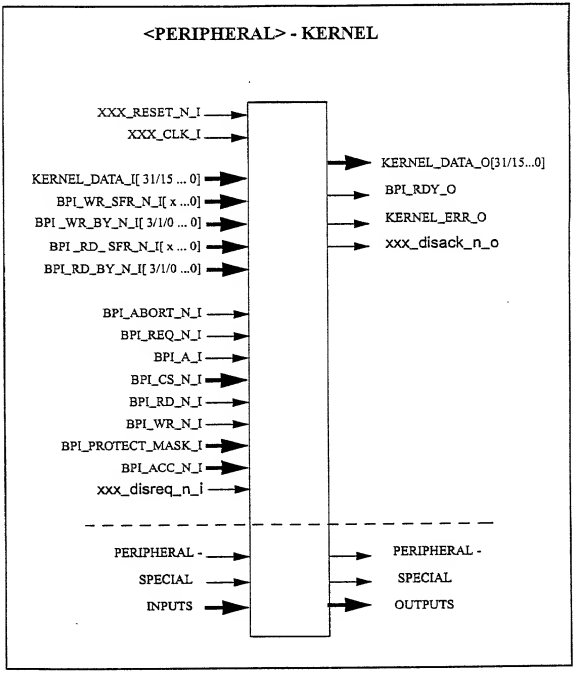


5.4. The read modify write FSM

The rmodw FMS rememberes the read part of a modify access and waites for the corresponding write access.



6. I/O signals from Peripheral Kernel



The signals have already been described above.
If no error indication from the kernel is required, two signals must be driven nonetheless by the architecture of the kernel:

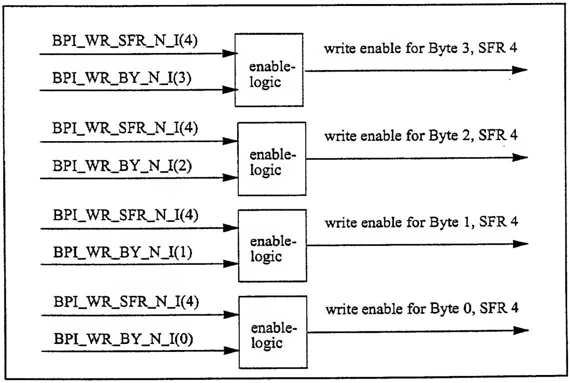
kernel_err_o <= ,0'; -- special error from kernel
xxx_disack_n_o <= xxx_disreq_n_i; -- always ok to disable clock

7. How you can use the port signals

7.1. Write enable logic

A register normally consist of one, two or four bytes. In order to write each byte separately the BPI provides a dedicated enable signal (BPI_WR_SFR_N_I(x)) for each SFR and a separate enable signal (BPI_WR_BY_N_I(3..0)) for each byte of the data bus.
For each byte of each register the write enable will be generated from this signals.
If the signal BPI_WR_SFR_N_I of the SFR and all signals of the BPI_WR_BY_N_I are low, then the whole register is enabled to store the new data.

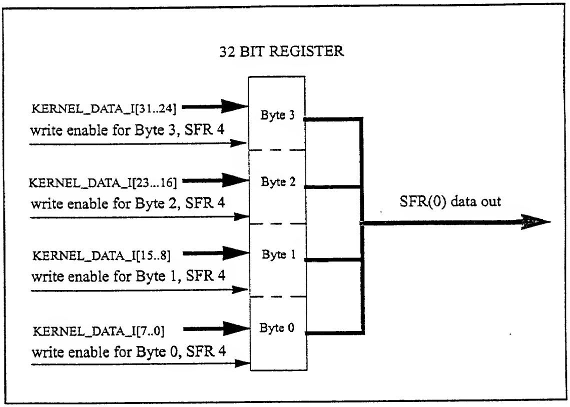
The figure below illustrates the proposed circuit



Caution!!
The BPI_WR_SFR_N_I[0 .. 3] and BPI_RD_SFR_N_I[0 .. 3] are reserved normally !!

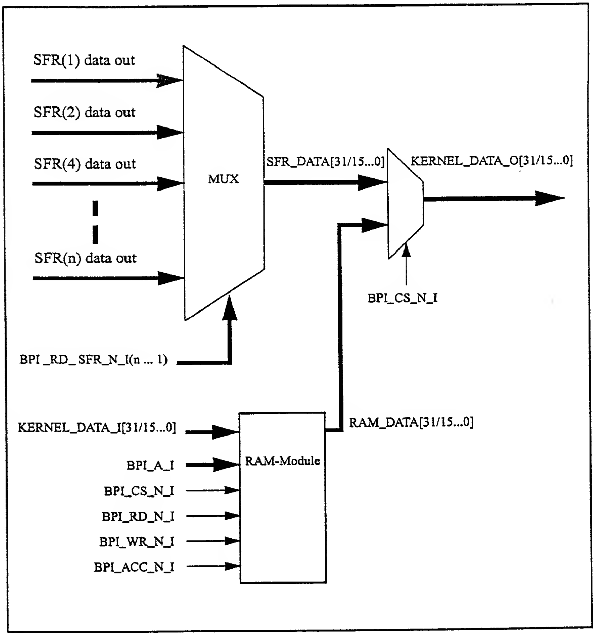
SFR(0) is the clock control register and is instantiated in the BPI.
SFR(1) is the peripheral input select register
SFR(2) is the identification number
SFR(3) is reserved for future use

The correlation between index and address is given by the order of the addresses in the addresslist in the bpi_package.
Note that only SFR(0) is implemented inside the BPI's architectures (clc-entity)


32 BIT REGISTER
KERNEL_DATA_I[31..24]
write enable for Byte 3, SFR 4
Byte 3
KERNEL_DATA_I[23...16]
write enable for Byte 2, SFR 4
Byte 2
SFR(0) data out
KERNEL_DATA_I[15..8]
write enable for Byte 1, SFR 4
Byte 1
KERNEL_DATA_I[7..0]
write enable for Byte 0, SFR 4
Byte 0

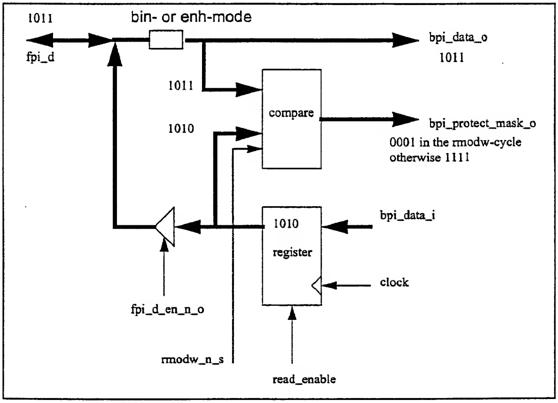
7.2. Multiplex output data

The next figure shows one possibility to propagate the output data of the kernel. Note that multiplexing RAM data and SFR data in two stages is only an example.

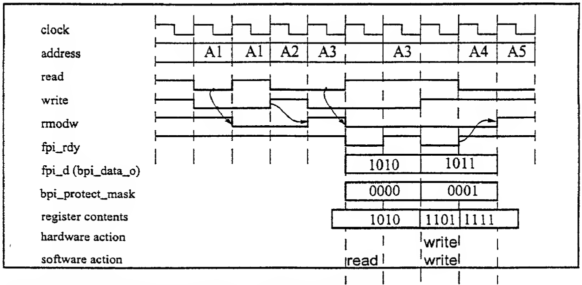


7.3. Read modify write cycle and protect mask

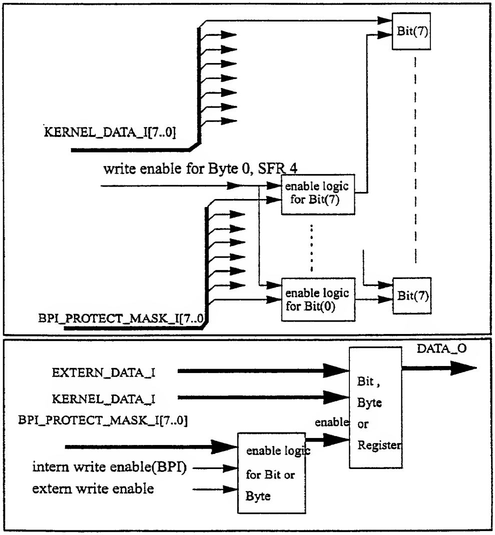
Read-Modify-Write is coded on the FPI bus and recognized by the BPI interface. The read modify write signal is not forwarded to the kernel.
As a result a signal RMODW_N_S is low active for the complete read modify write cycle on the FPI-BUS. As no other SFR may be read intermittingly the read data can be stored in the read register. A protection mask can now be derived by comparing the (locally stored) read data with the new data to be written. All bits not changed are given a mask value 0 if the RMODW_N_S signal is active.
In a normal write or read cycle, all bits of the mask are „1".
Protect mask bit equal to „1" means that the data bit must be written into the register.
Protect mask bit equal to „0" means that the data bit must not be written into the register.
This is illustrated by the next figures.



The bpi_data_i are only captured at the end of the read-cycle and remain unchanged by the write- and idle-cycles on the FPI bus. The protect mask is provided every time. Only register bits which can be modified by SW and HW need to use the protection mask.



The two figures below illustrate how to use these signals

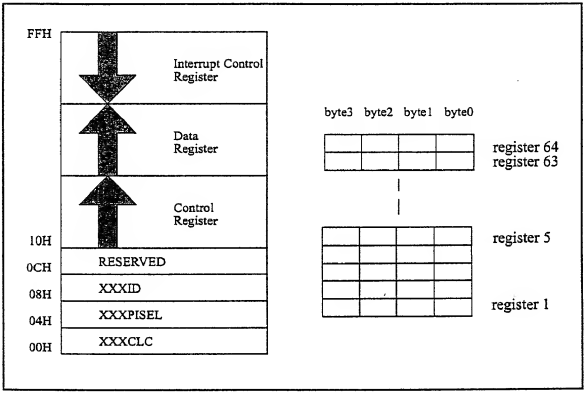


8. Peripheral Address Map

Each peripheral has n x 256 Byte register blocks or n x 64 32-Bit register. The register is selected with signals BPI_WR_SFR_N_I or BPI_RD_SFR_N_I . Four registers are fixed:

- Address 00H is the peripheral clock control register.
- Address 04H is the peripheral port input select register.
- Address 08H is the peripheral identification. No real register, hard coded.
- Address 0CH is reserved.

Control registers start at address 10H, followed by the data register.
The interrupt control register start at the address FFH decreasing order.



9. Usage of the fixed Registers

9.1. XXXID - REGISTER

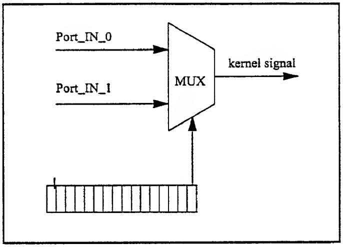
The XXXID is not a real register, it is hard coded and readable only.
Example :

```
        perid_s <= „10100111“;
                 or
        perid_c : std_ulogic_vector(8 downto 0) := „10100111“;
```

It is selected with bpi_rd_sfr_n_i(2).
It should be assured by appropriate dont_touch attributes that the ID is implemented on silicon as matel contacts. That allows cange of revision for metal redesigns.

9.2. XXXPISEL - REGISTER

The XXXPISEL register indicates which port shall be used for kernel inputs.
The register has to be implemented by the peripheral designer and is selected with bpi_wr_sfr_n_i(1). Every bit in this register controls one multiplexer.
Therefore, every external input of the kernel can be driven by one of two sources.
Example :
If XXXPISEL(0) = ‚0' then kernel signal <= Port_0 else kernel signal <= Port_1;



9.3. XXXCLC - REGISTER

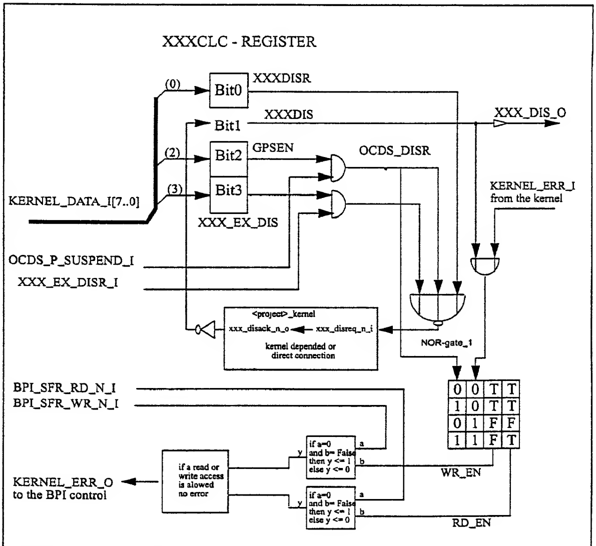
The state of the peripheral clock is controlled by a register bit "XXXDISR" located in the register XXXCLC of the peripheral core. This register XXXCLC is clocked with the bus clock to be able to switch the peripheral clock on again if it was off. If required by the peripheral's functionality switching off the clock can be prevented by the peripheral. The actual clock state will be shown by the state bit "XXXDIS" within the same register. The signal "XXX_DIS_N" which is a combinatorial combination of other clock control signals represents the actual enable state of the peripheral clock and is used to switch on/off the peripheral clock. The clock gating buffer is located in the Clock Gating block of a peripheral. The clock enable signals used to control the clock speeds are generated in the central Clock Generation and distributed to the Clock Gating block.
The BPI module rejects every FPI bus access with Error condition if the peripheral clock is switched off (signaled by the signal Kernel_err_o- driven by clc module, not peripheral kernel).

9.3.1. Coherence between XXXCLC and the OCDS_P_SUSPEND, XXX_EX_DISR

For On Chip Debugging support an additional signal OCDS_P_SUSPEND is intro-

duced to stop the peripheral clock for debugging if this function is enabled. If debugging mode is active the peripheral core rejects write access to registers connected to the peripheral clock by activating the signal Kernel_err. This causes the BPI module to reject FPI bus accesses with Error condition. Read accesses to registers of the peripheral clock domain are possible.
To be compatible with old products an XXX_EX_DISR signal is introduced to disable the peripheral clock.

For more information see also the **Peripheral Clock Strategy Specification**



clc data output is Bit(3 downto 0)

10. Forwarding

If a read access immediately follows a write cycle, it is normally not possible to read the previously written data in a zero waitstate access due to the required synchronisation. To resolve this problem no forwarding multiplexer is provided; instead a waitstate will be inserted if a read access is performed after a write access to the same register and without any idlestates in between !

11. Timing Diagrams

As there is no fixed relationship between bus clock and peripheral clock several different timing diagrams are provided.

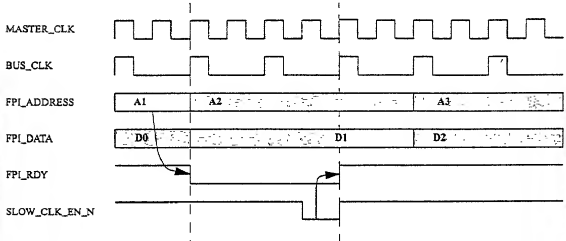
The first possibility is, that the bus clock is faster (bus_faster is ‚1') as the peripheral clock.

The second possibility is, that the bus clock is slower tahn or exactly as fast as the peripheral clock (bus_faster is ‚0').

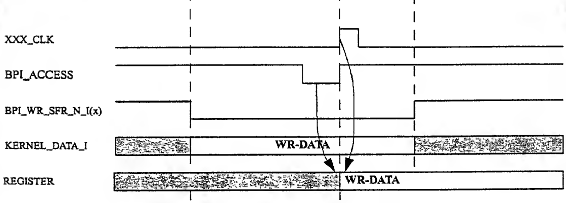
If it's necessary to implement a RAM inside the kernel the cycle must start with one waitstate. This is not implemented!

The BPI_ACC_N_I signal is used to avoid multiply writes to the same address due to to fast a peripheral clock. It will be driven low activ one clockcycle before the relevant rising edge of the peripheral clock. It is activated whenever a register must store the data in the write cycle or change the data after a destructive read cycle.

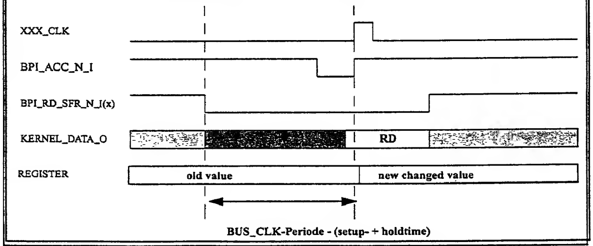
BUS CLK FASTER THAN XXX CLK

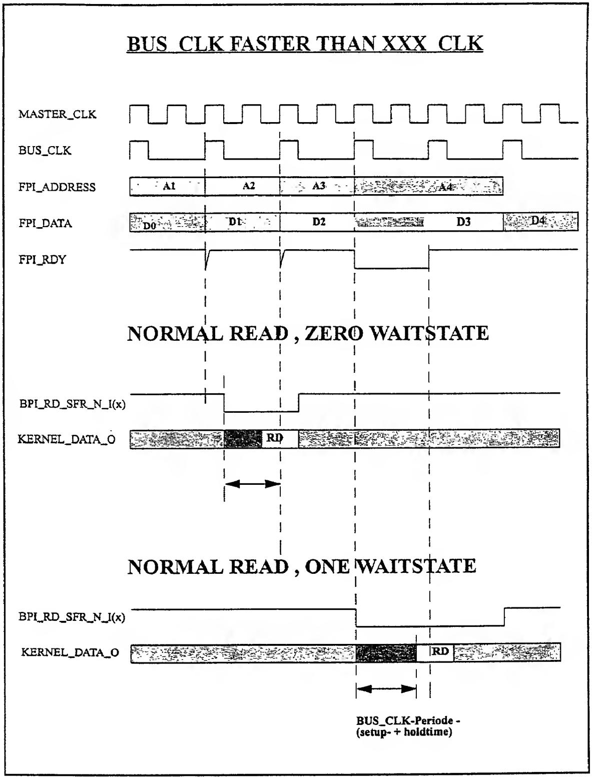


NORMAL WRITE , >= ONE WAITSTATE

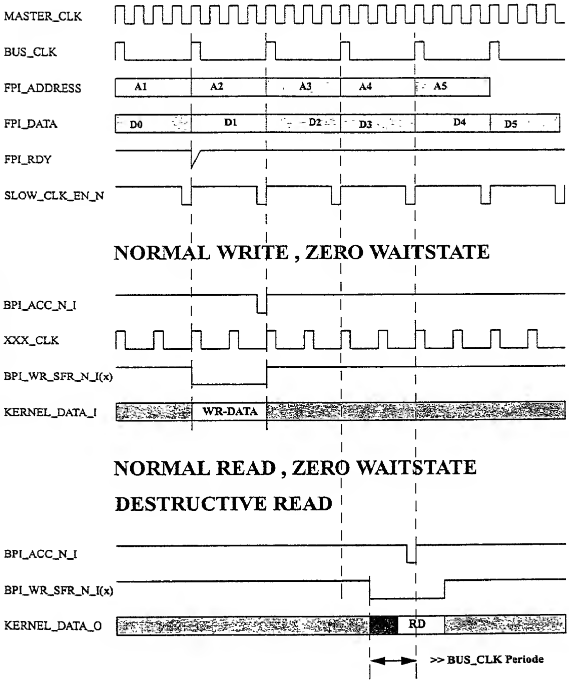


DESTRUCTIVE READ , >= ONE WAITSTATE

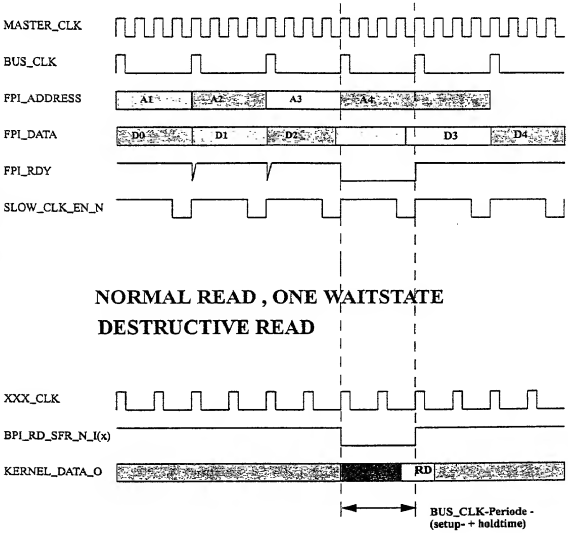



BUS CLK FASTER THAN XXX CLK
MASTER_CLK
BUS_CLK
FPI_ADDRESS
A1
A2
A3
A4
FPI_DATA
D0
D1
D2
D3
D4
FPI_RDY
NORMAL READ , ZERO WAITSTATE
BPI_RD_SFR_N_I(x)
KERNEL_DATA_O
RD
NORMAL READ , ONE WAITSTATE
BPI_RD_SFR_N_I(x)
KERNEL_DATA_O
RD
BUS_CLK-Periode -
(setup- + holdtime)

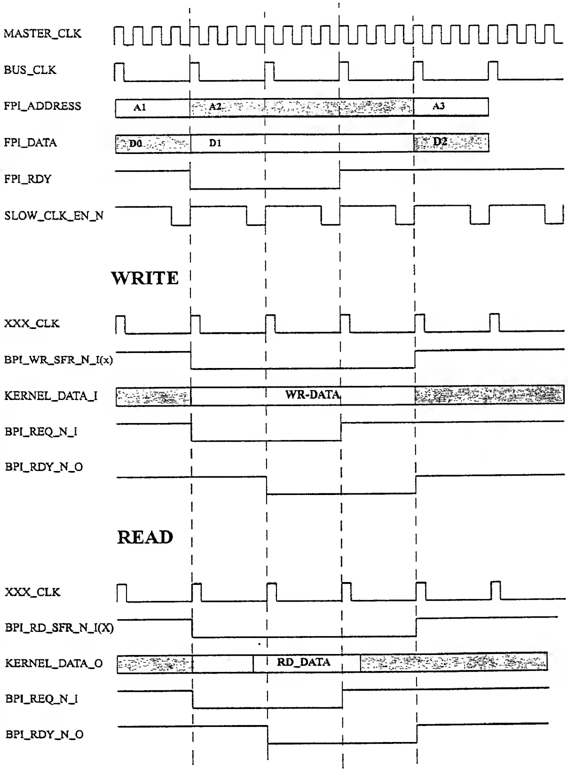
BUS CLK SLOWER OR EQUAL THAN XXX CLK



BUS CLK SLOWER OR EQUAL THAN XXX CLK

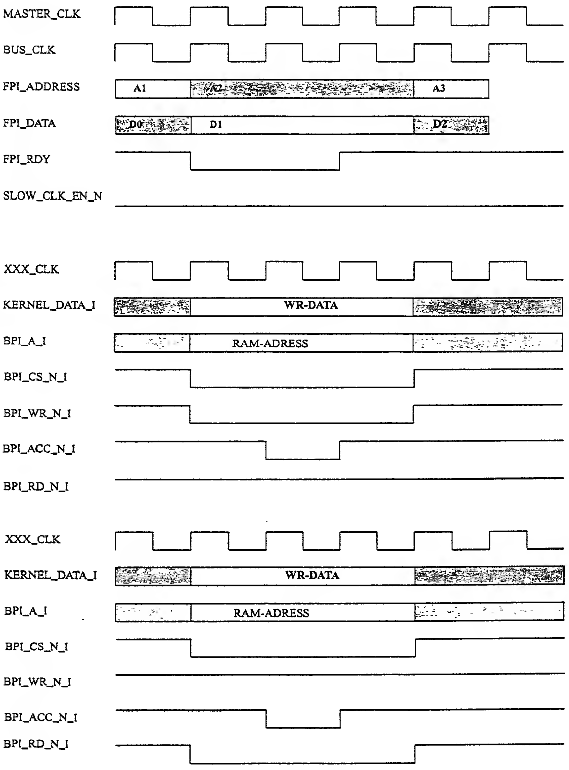


HANDSHAKE BETWEEN INTERFACE AND KERNEL

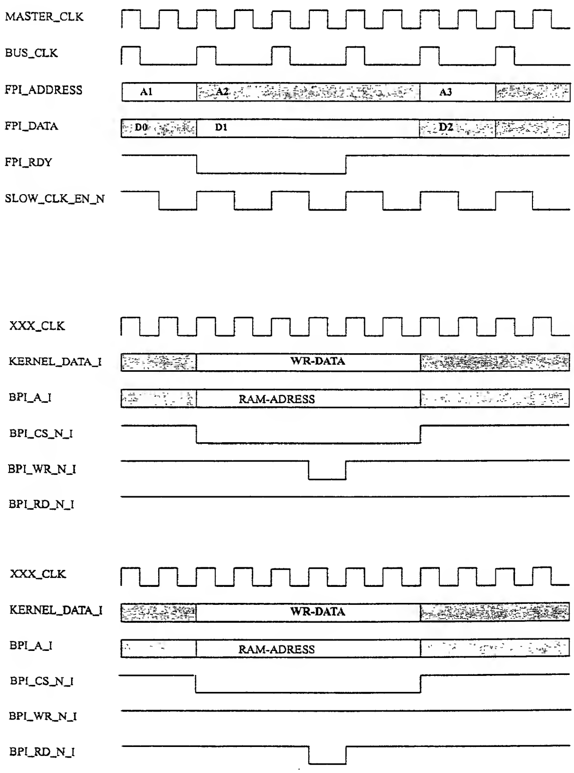


RAM INTERFACE (not implemented)

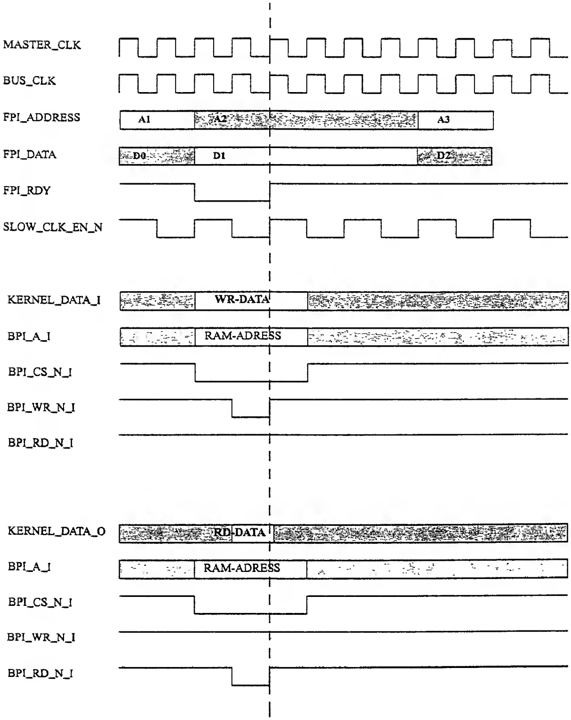
The current version is like a nomal access and provides the RAM signal but not in the timing below



RAM INTERFACE - BUS SLOWER THAN KERNEL



RAM INTERFACE - BUS SLOWER THAN KERNEL



12. Module Developers Test Cook Book

This chapter is a copy of Hermann Obermeir's paper regarding this topic; contact him for updates !

Platform peripherals are prepared for ATPG and for isolated module test. The following gives a brief overview what a module developer has to do.

12.1. Automatic Test Pattern Generation(ATPG)

12.1.1. Prepare setup files

e.g. GNUmakefile.setup
INSERT_SCAN_DESIGN := true
SCAN_CHAINS := <desired_number>

12.1.2. Run check test

- In Design Analyzer: Tools -> Test Synthesis: Check Design Analyzer
- you should have no violations

12.1.3. Insert Scan Registers and generate pattern

Use "ssemake insert_scan" to insert scan registers.
Build scan chains up to a maximum length of 100 flipflops.
serial inputs of the scanchains: **pdft_sci_0_i, pdft_sci_1_i,.....**
serial outputs: **pdft_sco_0_o, pdft_sco_1_o,....**
scan enable input: **pdft_scen_i**

12.1.4. Generate test Pattern

"ssemake atpg" You should achieve 100% fault coverage.
(If not make sure with a fault simulation of your module test pattern, that the untested faults are tested with your module test pattern)

12.1.5. Save Results

Save Results (Fault coverage,unvovered fauts)
Save constraints for the ATPG tool(set_test_hold, set_test_assume), if they have been used.

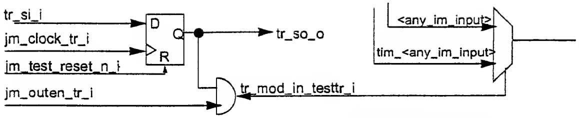
12.2. Preparation for isolated module test

The platform peripherals are tested in an isolated module test. The peripheral signals are made transparent at the chip boundary e.g. using multiplexers.
There are three categories of pins:

FPI bus: the FPI bus is made transparent via EBC/EBU
Alternate I/Os are made visible via normal prot functions
Outputs to other modules (here called intermoduleoutputs) are multiplexed in the ports
Inputs from other modules are multiplexed in the peripherals (intermoduleinputs)

12.2.1. Insert Testregister and Multiplex Inputs

Usually 1 test register bit is required. Use .... as an example
All InterModuleInputs are fed into a multiplexer and a testinput tim_.... is created, which will be connected to external pin on chip level



any_im_input = any_intermodule_input
tim = prefix for "test isolatd module"

tr_si_i, jm_clock_tr_si, jm_test_reset_n_i, jm_outen_tr_si are module inputs
tr_so_o is a module output. They will be connected at chip level

Further Cases:

- No test register: Some modules (SSC, ASC, IIC) have no dedicated intermodule inputs. In this cases the test register may be omitted. Such execptions have to be approved bye Georg Sigl, because they reduce testability.
- Addtional test register bits are necessary, if the module increases IDDQ-current (e.g. pullups, pulldowns) or the test register needs to be in a special mode for testing other peripherals (e.g. for clock generation)

12.2.2. Insert MISR

Insert Signature Registe Kann es je vorkommen, dass man das Signaturregister im Betrieb mitlaufen lassen moechte, dann braeuchten wir ein eigenes testregister Bit, ansosnter gemeinsam nit Eingansgsmultiplex

12.2.3. Insert MISR

You need a MISR, if your module has many inter module outputs (more than 5 to 10)

- Generate MISR using genbist (length is number of your intermodule outputs, but minimum 20
- add an asynchronous reset to the misr: jm_test_reset_i
- the bcode inputs of the MISR are test pins to your module: pdft_misr_bcode[1:0]
- the intermoduleoutputs are fed into the parallel MISR inputs
- the msb ofthe misr will be a test
- in test mode of the peripheral the MISR follows is controlled by the bcode test pins, in functional mode a synchronous reset is applied to bcode (bcode[1:0] ='10'
-

12.2.4. Select Module Pattern

The module pattern are used for a functional/performance test of the modul.
select a set of PDL files and store them.
Add a comment into your PDL file, where
If ATPG could not achieve 100% fault coverage make these

12.3. IDDQ Test preparation

IDDQ test vectors will be selected for platform products using Viewlogic/Sunrise. All peripheral have to fulfill the testability rules for this tool. Its not yet detemined, if we have to check every peripheral with Sunrise/testability checker.

12.4. Fault Coverage

12.5. Document Testability in Module Specification

Document Testability in your Module Specification. Use SSC as an example

13. Product Developers Test Cook Book

tbd

13.1. Preparations

13.1.1. Port Description

A description of the ports in computer readable format is necessary

13.2. Make Top Level Entity

13.2.1. Wiring of Test Register

13.2.2. Wiring of Scan Registers

13.3. Make Testbench

The necessay testbench contains

Appendix E

Title:
Macro Specification
DMUT Version 2.2

1 Introduction

1.1 Overview

The Transmit Data Management Unit (DMUT) provides direct data transfer from the shared memory to the internal Transmit Buffer (TB) with minimal host CPU intervention. The data buffers located in the shared memory are associated with one of the logical channels.

The on-chip Transmit Buffer (TB) requests data from the DMUT and after receiving it sends them to a Transmit Protocol Machine for protocol (HDLC, Ethernet, Frame Relay, ATM,...) processing.

A linked list of descriptors associated to each channel is located in the shared RAM and handled by the DMUT. The address generator of the DMUT supports full link list handling. The descriptors can be stored in separate memory location from the data buffers themselves allowing full scatter/gather assembly of packets. In order to have only one read access on the system bus for each descriptor, the current descriptor of each channel is hold on-chip. The interrupt vectors generated by the DMUT are transferred to a separate interrupt controller (DMUI). The DMUI will transfer the interrupt vectors of the complete device to the shared memory.

The number of channels is a flexible parameter of the DMUT macro. The DMUT can be used for multichannel devices (M256) as well as for high speed controllers with a reduced number of channels (DSCC4).

1.2 Features

- Data transmit from the shared memory to the central Transmit Buffer (TB)
- Support of linked list data structures located in the shared memory, consisting of a descriptor associated to a data section
- Independent channel configuration
- Forward interrupt vectors to the DMUI

1.3 System Integration

The DMUT has four interfaces:

- FPI Initiator Bus
- FPI Target Bus
- TB interface
- DMUI interface

The data are read from the FPI initiator bus and written to the TB interface. The descriptors are also accessed through the FPI initiator bus. The interrupt vectors are transferred on the DMUI interface bus. Each channel is configured independently (base address register, command register,...) via the FPI target bus.

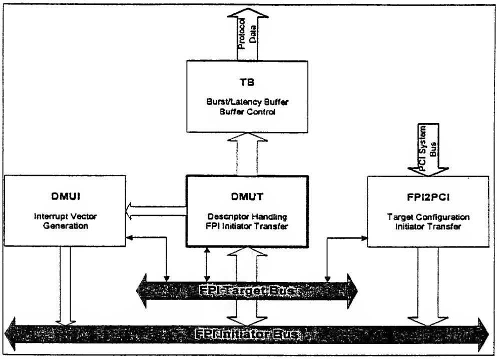


Figure 1
System Integration

1.4 Known Restrictions and Problems

The priorization on the FPI initiator is not an easy task.

DMUT should have priority if new data or a descriptor change is required in the middle of a frame in order to prevent underrun of the transmit buffer (and an abort of the frame). But at the beginning of a frame, the transfer is not time critical.

DMUT and DMUI should have an equal priority on the FPI initiator bus to prevent buffer underrun in the transmit buffer and buffer overflow in the IC.

2 Functional Description

2.1 Block Diagram incl. Clocking Regions

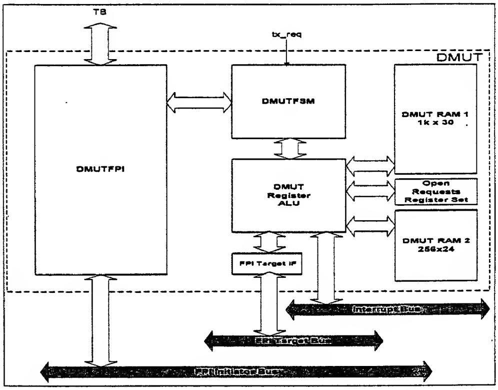


Figure 2
DMUT Block Diagram

The DMUT macro consists of 4 blocks, an open requests register set and 2 DMUTRAMs, that store the current transmit descriptor, the next transmit descriptor, the state of the channel, the interrupt mask and the interrupt queue number. The DMUTFPI contains a FPI initiator interface and is directly connected to the FPI Initiator Bus on one side and to the transmit buffer (TB) on the other side. This approach permits a fast transmission from the initiator bus to the transmit buffer without intermediate storage. The DMUTFSM handles the requests of the TB, controls the address and burst length calculation, initiates the data transfers from the shared memory to the TB and the updating of the current transmit descriptors by triggering the DMUTFPI. This block also controls the assembling of the interrupt vectors and writes them to the register block. There they will be stored until the request on the interrupt bus was granted and the interrupt information could be transmitted. The register block also stores information of the current channel available for the DMUTFSM and the ALU calculates addresses and burst lengths based on requested amount from the TB and the stored information in the DMUTRAM. The requests which could not have been responded so far are stored in the open requests register set. Configurational access from the CPU to the DMUT macro is possible via FPI Target Bus and FPI Target interface (refer to "SMIF Specification").

The complete DMUT block and all the interfaces are synchronous to the same CLK signal derived from the FPI initiator bus.

2.2 Normal Operation Description

The DMUT operates with a linked list of descriptors pointing to a data section. For each active channel a linked list is allocated in the external shared memory by the CPU. The current descriptor contains the pointer to the next descriptor, the start address of the data section and the reserved size of the data section.

For initialization the CPU programs the First Transmit Descriptor Address (FTDA) register, the interrupt mask, the interrupt queue and the Virtual Channel Specification command register (CSPEC_CMD) with the transmit init command(refer to chapter chapter 4.2). The byte swap mode (little/big endian - common for all channels) and the maximum burst length on the IFPI bus (also common for all channels) are programmend in global registers. The DMUT then fetches immediately all this information after an transmit init command and stores them in the chip internal RAM. When the transmit buffer first requests data for this channel the entire descriptor pointed by FTDA from the shared memory will be read. After storing the entire descriptor information in the on chip RAM, the DMUT will generate a Command Complete Interrupt (CMDC).

Table 1
Request Register (read access)

Bit	31	30 .. RLB+16	RLB+15... 16	15 . . CNB	CNB-1... 0
Function	DEL	Reserved	Requested Transfer Length RTL	Reserved	Channel number (CHN)

Table 2
Request Register (write access)

Bit	31	30 . . . RLB+16	RLB+15 . . . 16	15..CNB	CNB-1 . . . 0
Functi on	CI	Reserved	Served Transfer Length STL	Reserv ed	Channel number (CHN)

As long as the amount of empty locations in the transmit buffer is above the programmable threshold for a particular channel, TB will request a DMUT transfer. The DMUT will then read the request register at the TB-DMUT interface consisting of the DEL Bit (if set), the channel number (CHN) and the number of words (RTL) the TB requests from DMUT.

The DMUT reads the Transmit Descriptor corresponding to the channel number CHN out of its on-chip RAM and calculates the maximum number of bytes that can be read from the current transmit data section in the external memory based on checking the Byte Number (NO) field (part of the transmit descriptor), the amount of open request from former transfers, the requested transfer length and the maximum burst length.

The data transmission to the transmit buffer starts with the write of the Request Register consisting of the channel number and the served transfer length of the current transfer. Note that this amount of data might be smaller after initialization due to an end of frame (specific Frame End (FE) bit in the transmit descriptor) than the Requested Transfer length (RTL) . If this transfer contains a Frame End or a Transmit Abort Statusword the DMUT sets the Complete Indication bit (CI). During the transfer of a statusword the side-band signal dt_tx_stat is active. The transfer will be initiated by the DMUTFSM and executed by the DMUTFPI.

Depending on NO in the current transmit descriptor, the open requests and RTL one or several read accesses must be performed by the DMUT on the FPI initiator bus. The DMUT stops serving this request as soon as the requested amount of data was transfered to the TB, the DMUT transferred the maximum burst length (scalable in global register CONF3), when a Frame End bit (FE) in the current transmit descriptor is set or when the channel was aborted using an Transmit abort command (currently ongoing transfers will not be interrupted by this command). The difference between RTL and STL, called the open requests, which were not served in this cycle, will be stored in the open requests register set separately for each channel. When the TB requests new data for this channel CHN, the DMUT calculates the served transfer length STL out of the open requests, RTL, NO in the current transmit descriptor and the maximum burst length. Therefore Served Transfer Length might be more, equal or less than the Requested Transfer Length (RTL), but the maximum served length for one transfer cycle is 65

DWORDS (scalable in CONF3 to smaller maximum values for different system requirements) in the M256F application (max. for the FPI2PCI bridge - assuming that the data section contains enough data). The maximum amount of data the transmit buffer might request is 8K bytes.

In order to treat open request and requests from the transmit buffer equally, there is a continous switch between these 2 kinds of requests, if both are existent at the time. In case of the open requests, the DMUT scans through its open request register set and looks for open request which have not been served so far. If there are open requests for a channel, data transmission will be initiated. The procedure is the same as described above.

The DMUT will not branch to the next Transmit Descriptor of a channel when a FE Bit is set and all data were transferred, it serves a new request from the TB for another channel or looks for open requests in its register set. So open requests from other channels can get served faster, possible underruns might be avoided. The next Transmit Descriptor will be retrieved with the next data transfer of the channel.

During data transmission a channel might be reconfigured in order to handle more or less data. Therefore the reserved buffer size per channel in the transmit buffer has to be modified. The CPU turns this particular channel off using the transmit off command. The transmit buffer will free the allocated locations in the buffer for this channel and requests the DMUT to delete all stored open locations in the onchip RAM for the channel (DEL bit set in the request register). If this channel is just served by DMUT, data will be transferred to the transmit buffer, data have to be discarded there.

The DMUT will be informed about the tranmsit off command with the DEL bit in the request register set, the requested transfer length is invalid. The DEL bit will only be evaluated by the DMUT during reading the request register. The DMUT clears the open locations, sets the C-Bit in the Transmit Descriptor (if necessary) and generates a Command Completed Interrupt (CMDC) and a HI Interrupt (if HI bit is set in the Transmit Descriptor). Afterwards no more requests for this channel will be generated by the transmit buffer. After the CMDC Interrupt the CPU can set up the channel again (via INIT command) using another buffer size for the transmit buffer.

The DMUTFPI will perform split block read transfer (refer to FPI specification) to the FPI2PCI bridge. It writes data directly to the TB interface with minimum storage when the data are valid on the FPI intiator bus. If TB inserts wait states, DMUT can also insert wait states on the FPI initiator bus for the split response. Due to the latency on the PCI bus it may not be assumed that a requested data packet will be transfered in one cycle by the PCIFPI bridge, several smaller packets might be possible.

If the DMUT completes reading the data section associated with the Tx Descriptor, then the DMUT will set the Complete (C) bit in the Tx Descriptor if the CEN (Complete Enable) bit is set (refer to chapter chapter 2.3). Additionally an HI interrupt vector will be transferred to the DMUI if the HI bit is set in the Tx Descriptor. The DMUT will then branch to the next transmit descriptor. When a frame end is detected and all data have been transferred to the TB, the DMUT will add a frame end status word with/without data in order to inform the protocoll machine about the frame end.

The data transfer is controlled by the HOLD bit (CPU is owner of it), which is located in the transmit descriptor. If the HOLD bit is not set the DMUT will then branch off to the next descriptor. If the HOLD bit is set in the current Tx Descriptor this one is the last descriptor of the linked list available for the DMUT. The corresponding DMUT channel is deactivated as long as the CPU does not request an activation via the Virtual Channel Specification command register (refer to chapter chapter 4.2).

If the HOLD bit is detected in a descriptor and the Frame End bit is not set the DMUT will transfer all data of the belonging datasection. Afterwards the DMUT will set the C bit (if CEN is set) and generate an Hold Caused Transmit Abort Interrupt (HTAB) interrupt (the HI bit will be set in the interrupt vector as well when the HI bit is set in current transmit descriptor) in order to inform the CPU about the erroneous descriptor structure and writes an abort status word to the transmit buffer (protocoll machine info), since a complete frame could not be transmitted.

Then the DMUT reads the Current Transmit Descriptor once more. If the Hold Bit has been removed it will branch off to the Next Transmit Descriptor. When the Hold Bit is still set an internal Poll Bit will be set. Any further requests from the TB for this channel will not be responded, the number of requested data from the TB will be written in the open request register. In the meanwhile the CPU may issue a Transmit Abort Command for this channel. In this case, the DMUT will not repoll the old Transmit Descriptor and just issue a CMDC interrupt, since an abort indication was already sent to the protocoll machine. The DMUT continues with the descriptor pointed by FTDA.

If the HOLD bit and the Frame End Bit are set in the descriptor, the DMUT transfers the data of the belonging data section to the transmit buffer. At the end of the datasection it appends a frame end status word (trigger for protocoll machine). If the CEN bit is set, additionally the C-bit will be set in the current transmit descriptor, if the HI bit is set in the current transmit descriptor an HI Intr will be generated. When a new request for this channel appears (from tb or.from the open request registers) the DMUT repolls the FE/ HOLD descriptor. If the HOLD bit has been removed it will branch off to the next transmit descriptor. If the Hold bit is still set, the internal poll bit will be set and no HTAB INTR will be generated. Nor data neither a statusword will be sent to the transmit buffer in this case. Any further requests from the TB for this channel will not be responded, the number of requested data from the TB will be written in the open request registers.

Note that after a request from TB for a channel, which is on hold, the request register in the TB will not be written, since no data transfer will take place. Channels on HOLD will not be visible during scaning the open requests register set. When new data in the shared memory is available, the host CPU performs a Transmit Hold Reset command, the internal Poll Bit will be reseted.The status of this channel will be switched to ACTIVE in the onchip RAM, requests from the TB will be answered and open locations will be visible in the onchip open register set. In both cases the DMUT repolls the old HOLD

Descriptor, the HOLD Bit is suposed to be removed and it branches to the Next Transmit Descriptor. If the HOLD bit is not removed (erroneous programming by CPU) no action will take place. The transmit abort command is another way to switch a channel from HOLD to ACTIVE. In this case, the open requests will be made visible but the old descriptor will not be repolled and data transmission starts with the transmit descriptor pointed by FTDA .

The CPU will always issue a transmit hold reset command when it removes the Hold bit in a descriptor, no matter the DMUT has already seen the Hold bit or not.

The status of each channel (active or hold) is stored in the onchip RAM of the DMUT.

Although the DMUT works only 32-bit word oriented, it is possible to begin a transmit data section at an uneven address. The two least significant bits (ADD) of the transmit data pointer determine the beginning of the data section and the number of bytes in the first 32-bit word of the data section, respectively.

Table 3 shows how many bytes are valid in the first 32-bit word depending on the ADD bits for both little endian and big endian mode. Of course, the number of valid bytes depends on the length NO of the data buffer:

Table 3
Meaning of ADD in Little/Big Endlan Mode

ADD	Number of valid bytes	Little Endian				Big Endian			
00	4, if NO > 3	byte 3	byte 2	Byte 1	byte 0	byte 0	byte 1	byte 2	byte 3
01	3, if NO > 2	byte 2	byte 1	Byte 0	-	-	byte 0	byte 1	byte 2
10	2, if NO > 1	Byte 1	byte 0	-	-	-	-	byte 0	byte 1
11	1, if NO > 0	byte 0	-	-	-	-	-	-	byte 0

If the big endian byte ordering is programmed, the DMUT provides byte ordering swapping.

Note: The descriptor transfer is always a 32 bit access (no byte swapping).

2.3 Transmit Descriptor Definition

The transmit descriptor is initialized by the host CPU in the shared memory and is read by the DMUT. The address pointer to the first transmit descriptor will be stored in the onchip RAM, when requested to do so by the host via Transmit Init Command. The transmit descriptor is read entirely when the transmit buffer requests a data transfer for this channel and is stored in the on-chip memory. It will also be read entirely when branching from one transmit descriptor to the next. Therefore all information in the next descriptor must be valid when the DMUT branches to this descriptor.

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Byte Address																
0x00	FE	HOLD	HI	CEN	Reserved						Descriptor ID					
0x04	Next Transmit Descriptor Pointer															
0x08	Transmit Data Pointer															
0x0C	Rsvd	C	Reserved													

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Byte Address																
0x00	NO															
0x04	Next Transmit Descriptor Pointer															
0x08	Transmit Data Pointer															
0x0C	Reserved															

FE: Frame End, set by the host

It indicates that the current transmit data section (addressed by Transmit Data Pointer) contains the end of a frame. The DMUT will transfer data from this section to the transmit buffer (TB) and it checks the HOLD bit value. When HOLD=0 it branches to the next Tx Descriptor with the next request for this channel.

HOLD: Hold

It indicates whether the current descriptor is the last element of a linked list or not:

- HOLD=0: A next descriptor is available in the shared memory; after checking the HOLD bit stored in the on-chip memory the DMUT branches to next transmit descriptor.
- HOLD=1: The current descriptor is the last one that is available for the DMUT. The corresponding DMUT channel is deactivated for transmit direction as long as the CPU does not request an activation via the command register.

NO: Byte Number

The byte number defines the number of bytes stored in the data section to be transmitted. Thus the maximum length of data buffer is 65535 bytes. In order to provide dummy transmit descriptors NO = 0 is allowed in conjunction with the FE bit set. In this case (NO = 0) a HI INTR and/or the C-bit will be generated/set when the DMUT recognizes this condition. If NO = 0 without FE set two different cases are possible:

a) The last Transmit Descriptor contained a Frame End. Since no data were transfered for this transmit descriptor and the last frame was closed, no abort status word and no abort INTR will be generated.

If the hold bit is set (HOLD = '1'), a HTAB Intr will be issued in order to inform the CPU about the erroneous descriptor structure and the current Transmit Descriptor will be repolled immediately. If the HOLD bit is still set, the internal poll bit will be set. In order to move on, a Transmit Hold Reset (THR), a Transmit Abort or a Transmit Off Command can be issued. If the HOLD bit is removed after the repoll, the DMUT will branch to the next Transmit Descriptor.

If the HOLD bit is not set, the DMUT will branch to the next Transmit Descriptor

b) The last Transmit Descriptor did not contain a Frame End. An Abort Statusword will be generated and the corresponding TAB INTR issued.

If the HOLD Bit is not set in the current Transmit Descriptor, the DMUT branches off to the next transmit descriptor.

If the HOLD Bit is set, a HTAB Intr will be issued and this Transmit Descriptor will be repolled immediately. If the HOLD Bit is still set, the internal poll bit will be set. In order to move on, a Transmit Hold Reset (THR), a Transmit Abort or a Transmit Off Command can be issued. If the HOLD bit is removed after the repoll, the DMUT will branch to the next Transmit Descriptor.

The data bytes must be stored within the transmit data section according to the selected mode (little endian or big endian).

HI: Host Initiated Interrupt

If the HI bit is set, the DMUT transfers an HI interrupt vector to the interrupt controller after transferring all data bytes of the current data section to the internal transmit buffer.

Next Transmit Descriptor Pointer

This 32-bit pointer contains the start address of the next transmit descriptor, it has to be dword aligned. After sending the indicated number of data bytes, the DMUT branches to the next Tx Descriptor to continue transmission. The Tx Descriptor is read entirely at the beginning of transmission and stored in on-chip memory. Therefore all information in the next descriptor must be valid when the DMUT branches to this descriptor.

This pointer is not used if a Transmit Abort command for this channel is detected while the DMUT still reads data from the current transmit descriptor. In this case the First Transmit Descriptor Address (FTDA) register is used as a pointer for the next transmit descriptor to be branched to.

Transmit Data Pointer

This 32-bit pointer contains the start address of the transmit data section. Although the DMUT works long word oriented, it is possible to begin transmit data section at byte addresses.

C: Complete

This bit is set by the DMUT if

- it completes reading data section normally
- it was aborted by a transmit off or transmit abort command.

The Complete bit releases the descriptor (owner bit).

CEN: Complete Enable

This bit is set by the CPU if the complete bit mechanism is desired:

- CEN=0: the DMUT will NOT update the Tx Descriptor with the C bit. In this mode the use of the HI interrupt is recommended.
- CEN=1: the DMUT will set the C bit.

Descriptor ID

This field is read by the DMUT and written back in the corresponding interrupt status vector for HI , TAB or HTAB interrupt. This value provides a link between the interrupt vector and the Tx Descriptor, to be used by the software.

2.4 Interrupt Vector Definition

The DMUT generates 2 groups of interrupts, channel command related interrupts and interrupts which are related to the data transfer of the DMUT for a particular channel. These 2 groups will be distinguished by the Interrupt ID. Additionally the channel number and specific bits are part of the interrupt vector. The command interrupts will be written to a designated queue with a fixed length. Channel interrupts can be written to 8 different queues with variable length, the specific queue for each channel will be programmed during initializiation of the DMUT.

For Channel Interrupts the Descriptor ID value is written back to the corresponding interrupt vector. This provides a link between the interrupt vector and the descriptor, to be used by the software. The interrupt vectors are transferred to the interrupt controller and have following format:

Table 4
Interrupt Vector Format (Channel Interrupt Vector)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
1	0	1	1	1	Q2	Q1	Q0	0	0	Descriptor ID					

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
HI	TA B	0	HT AB	0	0	0	0	Channel No.							

Command Interrupt Vector

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	CM DC

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	Channel No.							

Channel number

Identifies the channel where the interrupt occurred

Descriptor ID

Identifies the transmit descriptor, which generated the interrupt

HI: Host Initiated Interrupt

If the HI bit in the transmit descriptor is set, an Interrupt Vector with this bit set is generated when DMUT finishes the old transmit descriptor and branches to the next transmit descriptor.

Q2,Q1,Q0: Queuenumber

Identifies the number of the interrupt queue for the channel in the shared memory (needed by the interrupt controller - not valid for the Command Intr.)

TAB: Transmit Abort

Issued if the DMUT detects a transmit abort command or an erroneous transmit descriptor (FE=0 und NO=0) and a frame was not transmitted completely for this channel.

HTAB: Hold Caused Transmit Abort

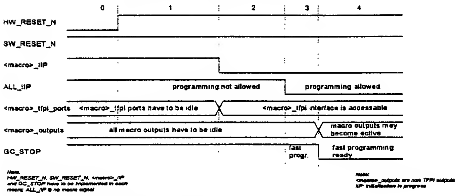
Issued if the DMUT retrieves a transmit descriptor where HOLD =1 and FE = 0. The interrupt will be generated after the data section was transferred completely.

CMDC: Command Completed

CMDC = 1 identifies a successful channel command performed by the host CPU. The DMUT is ready for a new command by the CPU for this channel.

2.5 Reset Behavior

Initialisation of M256F (macro view)


0
1
2
3
4
HW_RESET_N
SW_RESET_N
<macro>_IIP
ALL_IIP
programming not allowed
programming allowed
<macro>_tfpi_ports
<macro>_tfpi ports have to be idle
<macro>_tfpi interface is accessable
<macro>_outputs
all mecro outputs heve to be idle
macro outputs mey become ective
GC_STOP
fast progr.
fast programming ready
Note.
HW_RESET_N, SW_RESET_N, <macro>_IIP and GC_STOP have to be implemented in each macro; ALL_IIP @ no macro signal
Note:
<macro>_outputs are non TFPI outputs
IIP: initialisation in progress

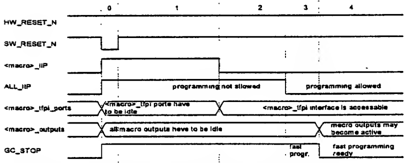
0: HW_RESET_N is active and deactivates SW_RESET_N; all macros are reset; GC_STOP set to '1'

1: HW_RESET_N inective => each macro executes the macro specific initialleation of rams

2: some macros are ready with initialisation of rams, some not (ALL_IIP is a or-function of all <macro>_IIPs). The "ready mecros" could be accessed by TFPI interface, but software programming is not allowed as long as ALL_IIP is active

3. all macroe are ready with initialisation => ALL_IIP becomes inective; software polls the ALL_IIP bit after HW_RESET_N; if ALL_IIP = 0 then software programms all macros (in e default "fast programming mode", GC_STOP = 1; reset value of GC_STOP is '1', but can be raset at any time)

4: after fest programming macro outpute (non TFPI) may become active and macro has to react on non FPI inputs



0: SW_RESET_N becomes active; all macros (registers) are reset; GC_STOP is active

1: SW_RESET_N inactive => each macro executes the macro specific initialisation of rams

2: some macros are ready with initialisation of rams, some not (ALL_IIP is e or-function of all <macro>_IIPs). The "ready macros" could be accessed by TFPI interface, but software programming is not allowed as long as ALL_IIP is active

3: all macros are ready with initialisation => ALL_IIP bacomes inactive, software polls the ALL_IIP bit efter SW_RESET_N; if ALL_IIP = 0 then eoftware programme all macros in e "fast programming mode", GC_STOP = 1

4· after fast programming macro outputs (non TFPI) may bacome active and each macro has to react on non FPI inputs

All internal registers and functions are initialized to known states (RESET state).

In addition to this global reset, each channel can be turned off or aborted via the command register (refer to chapter 4.2.1).

2.6 Functional Test Description

2.7 Production Test Description

3 Macro Interfaces and Signal Description

All signals are active high until otherwise specified. Active low signals are designated by "_N" appended to their names. To make the design as re-usable as possible, a bus signal whose width is application dependent is specified with one of the following parameters:

Table 5
Application specific Parameters

Parameter name	Bus Type	Typical value (bits)
CNB	Channel Number Bus	8
DBB	Data/Status Bus	32
BLB	Burst Length Bus	6
RLB	Request Length Bus	13

3.1 Signal Description

In the following sections, "Flexible Peripheral Interconnect (FPI) Bus compliant" means that the specified bus uses a subset of the FPI features and satisfies the basic address and data cycle. Not all FPI signals are implemented because default values are sufficient for the application i.e. they can be coded as constants in the hardware. Refer to the FPI bus specification for details of the complete bus.

The following tables list the FPI-Bus signals and two additional out-band signals:

- dt_tx_stat to indicate if transferred word is status instead of pure data. This signal is only active during the transfer of a statusword (1 clock cycle).
- tx_req_n to indicate that the Transmit Buffer Action Queue (TBAQ) in the TB is not empty.

Table 6
Macro Interfaces and Signal Description

Symbol name	I/O	Function
Control Signals		
sysclk	I	system clock
reset_n	I	Reset
scanmode	I	Scanmode
gc_stop	I	global stop signal, used for fast RAM programming by the CPU
dt_iip	O	dmut macro initialization in progress, when active the DMUT macro initializes its internal RAMs
Transmit Buffer (TB) interface		
dt_tx_rd_n	O	Read Control (for request register)
dt_tx_wr_n	O	Write Control
dt_tx_a	O	1 bit address bus 0 => status port. 1 => data port.
dt_tx_d[DBB-1:0]	O	Output data/status from DMUT to TB controller
tx_d[DBB-1:0]	I	Input data from TB (request register)
tx_rdy	I	End of data transfer indication. 0 => DMUT should insert wait state. 1 => transmit buffer will finish transfer during this clock cycle.
dt_tx_stat	O	Current transferred word is statusword
tx_req_n	I	Service request from transmit buffer to DMUT controller

Table 6

Macro Interfaces and Signal Description (cont'd)

Symbol name	I/O	Function

Interrupt Controller (DMUI) Interface

dt_int_req_n (**)	O	Interrupt Bus Request Line
ic_dt_gnt_n(**)	I	Interrupt Bus Grant Line
dt_int_d[DBB-1:0]	O	Interrupt Vector
dt_int_d_en[7:0]	O	Output Enable Bus for data

FPI Target Interface

tfpi_a[8:2]	I	Address bus.
tfpi_d[DBB-1:0]	I	Data bus. Active during data phase.
tfpi_rdy	I	Ready Input, other target will finish datat cycle in the clock cycle (tfpi_rdy_n = 1) or it will insert wait states

tfpi_wr_n
tfpi_rd_n
	I
	I
		Read/Write control. Following codes are defined:
		WR_N = 1; RD_N = 1 => NOP
		WR_N = 0; RD_N = 1 => data written to DMUT
		WR_N = 1; RD_N = 0 => data read from DMUT

tfpi_vc_sel_n	I	DMUT Slave select (registers of virtual channel specification) 0 => address is valid 1 => address is not valid
tfpi_vg_sel_n	I	DMUT Slave select (global registers) 0 => address is valid 1 => address is not valid
dt_tfpi_d[DBB-1:0]	O	Data bus output. Active during data phase.
dt_tfpi_d_en[7:0}	O	Data bus output enable bus (one enable line drives 4 data lines)
dt_tfpi_rdy	O	End of transfer indicator: 0 => Master should insert wait states 1 => DMUT will complete transfer in this cycle
dt_tfpi_rdy_en	O	Output Enable signal for Ready Output

Table 6
Macro Interfaces and Signal Description (cont'd)

Symbol name	I/O	Function
FPI Initiator Bus		
ifpi_gnt_n	I	DMUT Initiator Grant Line
ifpi_opc[3:0]	I	Operation Code (Input for split read response)
ifpi_d[DBB-1:0]	I	Data Bus
ifpi_sel_n	I	DMUT select signal, used for split block response
ifpi_ack[1:0]	I	Slave Response code
ifpi_rdy	I	Slave (PB) on data bus will be able to finish transaction in current clock cycle
dt_ifpi_req_n	O	DMUT Initiator bus request line
dt_ifpi_rd_n	O	Read Control Output
dt_ifpi_rd_n_en	O	Read Control Output Enable
dt_ifpi_wr_n	O	Write Control
dt_ifpi_wr_n_en	O	Write Control Output Enable
dt_ifpi_abort_n	O	Abort signal
dt_ifpi_abort_n_en	O	Abort Output Enable
dt_ifpi_opc[3:0]	O	Operation Code
dt_ifpi_opc_en	O	Operation Code Output Enable
dt_ifpi_tc[5:0]	O	Transfer Count (max. burst size 64dwords)
dt_ifpi_tc_en[1:0]	O	Transfer Count Output Enable Bus (one enable line drives 4 data lines)
dt_ifpi_a[DBB-1:0]	O	Address Bus
dt_ifpi_a_en[7:0]	O	Address Output Enable Bus
dt_ifpi_d[DBB-1:0]	O	Data Bus
dt_ifpi_d_en[7:0]	O	Data Bus Output Enable Bus
dt_ifpi_tag[3:0]	O	Tag Bus (used to transfer the master ID for Split Block Transfers)
dt_ifpi_tag_en	O	Tag Bus Output Enable

Table 6
Macro Interfaces and Signal Description (cont'd)

Symbol name	I/O	Function
dt_ifpi_rdy	O	DMUT will be able to finish transaction in the current clock cycle
dt_ifpi_rdy_en	O	Ready Output Enable

(**) These signals connect DMUI and the arbitration logic on the DMUI bus.

3.2 Data Flow and Functional Timing

3.2.1 FPI Master Bus

Refer to the SIEMENS FPI Bus for Munich-Macros Specification

3.2.2 FPI Slave Bus

Refer to the SIEMENS FPI Bus for Munich-Macros Specification

3.2.3 Interface between DMUT and Transmit Buffer

The DMUT interface to the TB is based on a bidirectional FPI like bus. Because only one operation code is defined (Single Word Transfer), an OPC signal is not necessary.

The tranmsit buffer requests data and the DMUT either serves the request (requested length < served length, requested length = served length, requested length > served length) or not (channel on HOLD). Before the DMUT starts the data transfer to the TB it writes the amount of data, the channel number and the CI (if necessary) to the request register (refer to chapter 2.2).

The select signal SEL_N is implicitly active and not physically present. DMUT sees TB as a request register at address 0 and a FIFO data port at address 1. Only 1 address bit (dt_tx_a) is defined on the bus.

Two out-of-band signals are required

1. tx_req_n to indicate that TB FIFO has empty locations.
2. dt_tx_stat to indicate the currently transmitted word is status word instead of pure data.

When the signal tx_req_n is active and the DMUT is in idle state, the DMUT initiates an address cycle by asserting address 0 (request register). During the data cycle, TB returns the request register, which describes the port status and asserts tx_rdy.

If the channel is not on HOLD, the DMUTFSM calculates the Served Transfer Length and writes STL together with the channel number and the CI bit (if necessary) into the request register (address 0). The DMUTFSM initiates the transfer writing the start address and the amount of data to the DMUTFPI. The DMUTFPI block requests the data via FPI Initiator bus from the shared memory and then transfers STL words with either individual or overlapped cycles. In best case, a burst cycle can occur without wait states but internally, the PMT interface of the transmit buffer has higher priority. In the following figure, a complete transfer (read request register, write request register, write data) with collision with the PMT interface (2 wait states) is shown. Note that the TB also can insert wait states during the writing of the request register (not shown in the diagram!).

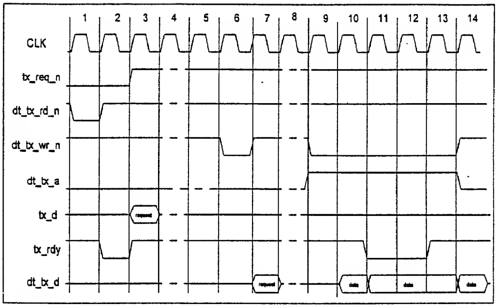


Figure 3
Complete transfer cycle

The efficiency of the data transfers improves with burst length. The peak throughput (not sustainable) can approach 1 word/clock with long bursts (theoreticly; M256F 1 word/3 clocks). However it is ultimately limited by the PMT interface access of the internal transmit buffer RAM. Each word the protocoll machine requests during a TB-DMUT data cycle may results in 2 additional wait states on tx_rdy.

The amount of transferred data depends on the requested amount of data by the TB and the open requests for this channel. The DMUT checks whether the current data section contains enough data. If not the DMUT will first transfer the rest of the current data section (write request register with STL = rest of current data section, transfer data). If the FE bit is not set in the Current Transmit Descriptor the DMUT will branch to the next Descriptor and transfer the rest of requested transfer (write request register, transfer data). **Figure 4** illustrates the 2 step transmission. Note that between writing the request register and the data transfer itself a couple of clock cycles may elapse since the DMUTFPI requests the data out of the shared memory via the FPI initiator bus.

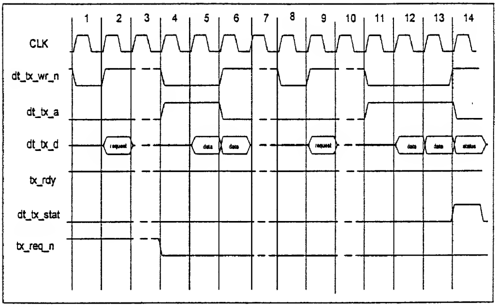


Figure 4
Transfer to TB from different datasection of TX Descriptor

For the last word of a frame transferred to TB or if a transmit abort occurred because HOLD=1 and FE=0 or NO=0 and FE=0 are detected in the transmit descriptor or a transmit abort is performed in the middle of a frame via the command register, the DMUT transfers a status word and activates the dt_tx_stat line (at the beginning of the cycle during writing the request register the CI bit will be set - status word indication). In this case the word has the following format:

Table 7
Status word

Bit	31	30	29..26	25	24	23 16	15. ...8	7 0
Function	FE	TAB	Reserved	BE1	BE0	Byte2	Byte1	Byte 0

BE[0..1]=00: Byte 0-2 are invalid, the word contains only status information

BE[0..1]=10: Byte 0 is valid, Byte 1-2 are invalid user data.

BE[0..1]=01: Byte 0-1 are valid, Byte 2 is invalid user data.

BE[0..1]=11: Byte 0-2 are valid

TAB (Transmit Abort) allows the protocol machine to send an abort sequence for the frame. FE allows the protocol machine to close the transmission of a frame (e.g. with CRC and flag).

3.2.4 DMUI Interface

The DMUT transfers the interrupt vectors to the DMUI (Interrupt controller) which will arbitrate the FPI initiator bus and transfer the vector in the corresponding interrupt queue in the shared memory. The interface between DMUT and DMUI is a bus shared by all the blocks generating interrupts. Therefore the transfer starts with an arbitration cycle and it might take some cycles until the request is granted, minimum latency is 1 clock cycle. Once the bus is granted to DMUT, DMUT deactivates its request in the next cycle and writes the vector on the dt_int_d data bus in the same cycle (because no address cycle is needed, additionally dt_int_d_en will be asserted). Since the DMUI will not insert wait states, a RDY signal from the DMUI is not necessary.

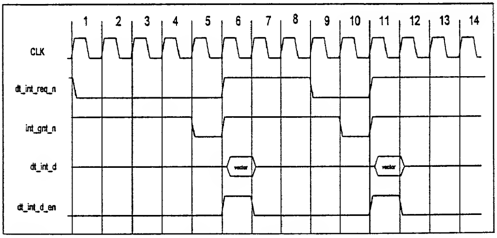


Figure 5
Interrupt Vector Transfer to the DMAI

3.3 Macro Functional Test

3.4 Macro Production Test

4 Register Description

4.1 Register Overview

The DMUT macro will be used in a system. Within the system it will happen that certain configuration information will be used by more than one macro. System programming would be very ineffective if these configuration information will be stored at different location in different macros. That's why a broadcast programming will be introduced. In order to set up channels a virtual channel specification, consisting of a set of registers, selected by the signal tfpi_vc_sel_n, will be programmed. Global information will be stored in a set of registers selected by the signal tfpi_vg_sel_n. Every macro implements only the registers or parts of the registers which contain necessary information.

All registers will be accessed via the target FPI bus, inside the macro the SMIF-BPI interface will control these accesses.

Essential for all slave register are fast accesses in write direction (i.e. no PCI wait states) and that a mechanism is provided to ensure that accesses are completed internally before the next command is given (i.e. no erroneous overwrite of data).

Read accesses are non critical, but a debug read back of all registers (virtual and standard) must be ensured;

Test mode accesses are non critical and can deliver deliberate results; i.e. read back RAM contents

The following chapter describes a subset of registers, which are necessary for programming the DMUT macro. The reset values are the driven values from the DMUT, in the system environment they might be different (non implemented bits by the DMUT might vary)

4.2 Detailed Register Description of the Virtual Channel Specification (selected by tfpi_vc_sel_n)

4.2.1 Virtual Channel Specification Command (CSPEC_CMD)

Access	: read/write
Address	: 00_H
Reset Value	: 00000000_H

31 24	23 16
CMD_XMIT(7:0)	rsvd

15	0
00	CHAN(7:0)

The reserved bits are implemented by others macros within the system.

The following commands will be supported:

Transmit Init
This command will cause the initialization of the channel CHAN(7:0). The DMUT will store First Transmit Descriptor Address (CSPEC_FTDA register) in its on-chip RAM. With the next request for this channel from the transmit buffer the entire descriptor pointed by the CSPEC_FTDA will be retrieved and afterwards a Command Complete Interrupt (CMDC) will be generated.

Transmit Off
The DMUT does not interprete this command, it will be done by the transmit buffer and the transmit buffer sets the DEL bit. The DMUT macro will see the DEL bit set for this channel in the request register of the transmit buffer. After reading this request register containing the DEL bit all open requests for this channel will be removed and and a Command Complete Interrupt (CMDC) will be generated. The C-bit will be set in the old Transmit descriptor and a HI INTR generated, if necessary (CEN, HI).

Transmit Abort
If this command is detected the First Transmit Decriptor Address (CSPEC_FTDA Register) will be stored. With the next request for this channel the current transmit descriptor will be suspended.

If the frame was correctly closed (FE), the new transmit descriptor, pointed by FTDA, will be retrieved.

If a frame was not sent completely (no FE) the DMUT will send a transmit abort status word to the transmit buffer and generate a transmit abort interrupt (for transparent mode these 2 actions will always be executed). The C-bit will be set in the old Transmit descriptor and a HI INTR issued, if necessary (CEN, HI).) Then the new transmit descriptor, pointed by FTDA, will be retrieved.

As soon as the entire descriptor is stored in the onchip memory, a Command Complete Interrupt will be generated.

When the DMUT has seen the HOLD bit for a channel (this channel is on hold), also an Transmit Abort Command can be issued, with the next request for the channel the DMUT will branch immediately to the transmit descriptor pointed by the CSPEC_FTDA without repolling the old one

Transmit Hold Reset

Always when the CPU removes the HOLD bit in the transmit descriptor chain of the channel additionally it has to write this Transmit Hold Reset Command (THR). Internal action refer to description of the polling mechanism (chapter 2.2). When the DMUT repolls the old transmit descriptor and the hold bit is still set (erronoeus programming of the software), no special action takes place.

Transmit Debug

All register belonging to the Virtual Channel Spec will be read with this command. The DMUT provides the contents of all the registers (CSPEC_FTDA only valid after the first request for the channel, contains the current transmit descriptor) described in this section.

Issuing the Transmit Hold Reset (THR) Command will be accepted immediately without acknowledgement. After executing one of the other commands (init, off, abort) by the DMUT macro a Command Complete (CMDC) will be generated. The CPU may not write another command for this particular channel into the Command Register before the CMDC INTR occured. The commands are coded in the following way (the commands transmit idle and transmit update do not have an impact for the DMUT, they will be used by other macros in the system:

Command Table Receive:

31	30	29	28	27	26	25	24	
Rsvd	Update	Idle	Debug	HOLD RESET	ABORT	OFF	INIT	function
0	0	0	0	0	0	0	1	transmit init
0	0	0	0	0	0	1	0	transmit off
0	0	0	0	0	1	0	0	transmit abort
0	0	0	0	1	0	0	0	transmit hold reset
0	0	0	1	0	0	0	0	transmit debug
0	0	0	0	0	0	0	0	transmit nop
0	0	1	0	0	0	0	0	transmit idle
0	1	0	0	0	0	0	0	transmit update

4.2.2 (Virtual) Channel Specification Buffer (CSPEC_BUFFER)

Access	: read/write
Address	: 20_H
Reset Value	: 00000000_H

31	
TQUEUE(2:0)	rsvd

15				0
rsvd	0 rsvd 0	rsvd	0	rsvd

Note: The reserved bits are implemented by other macros within the system.

TQUEUE: The generated transmit channel interrupts will be sent to this interrupt queue (command INTR will be sent to the command queue)

4.2.3 (Virtual) Channel Specification FTDA (CSPEC_FTDA)

Access	: read/write
Address	: 28_H
Reset Value	: 00000000_H

31	0
FTDA(31:0)	

FTDA(31:0): First Transmit Descriptor Address (dword aligned)

4.2.4 (Virtual) Channel Specification Interrupt Mask/Queue (CSPEC_IMASK)

Access	: read/write
Address	: $2C_H$
Reset Value	: 00000000_H

31							Tran smit								
0	TAB	0	HTAB	0	0	0	0	rsvd	rsvd	0	0	0	0	0	TCC

15							Rec eive								0
0	rsvd	rsvd	rsvd	rsvd	rsvd	rsvd	rsvd	rsvdP	rsvd	rsvd	rsvd	0	rsvd	0	rsvd

Note: The reserved bits are implemented by other macros within the system.

Interrupt Generation Mask: These bits correspond to the respective channel interrupts, if set to '1', the corresponding interrupt will not be generated by the macro

4.3 Detailed Register Description of the Global registers (selected by tfpi_vg_sel_n)

4.3.1 Configuration Register 1 (CONF1)

Access	: read/write
Offset Address	: 40_H
Reset Value	: 00000000_H

31GC						GC									
rsvd	0	0	0	0	0	rsvd	0	0	0	rsvd	[illegible]	[illegible]	rsvd	[illegible]	[illegible]

15

[illegible]	[illegible]	rsvd	rsvd	[illegible]	[illegible]	[illegible]	[illegible]	rsvd	rsvd	rsvd	0	rsvd	rsvd	LBE	rsvd

Note: The reserved bits are implemented by others macros within the system.

LBE: Little/Big Endian Byte Swap: '0': Little Endian ; '1' : Big Endian

4.3.2 Configuration Register 3(CONF3)

Access	: read/write
Offset Address	: 48$_H$
Reset Value	: 00090000$_H$

31

0	0	0	0	0	0	0	0	0	0	0	0	TPBRSTL(3:0)			

15

0	0	0	0	0	0	0	0	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]

Note: The reserved bits are implemented by others macros within the system.

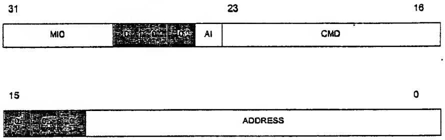
TPBRSTL(3:0): Transmit packet burst length
Sets the maximum PCI burst length for tramsit packet
data

0000	1 DWORD
0001	4 DWORDs
0010	8 DWORDs
0011	12 DWORDs
0100	16 DWORDs
0101	24 DWORDs
0110	32 DWORDs

0111	40 DWORDs
1000	48 DWORDs
1001	64 DWORDs

4.3.3 Test Command Register (TAC)

Access : write/read
Offset Address : 58_H
Reset Value : 00000000_H



MID: Macro ID Code (Bit 31:28 = 0110 for DMUT)
AI: Auto Increment Function
Address: internal address
CMD: Command (select of ram and register)

CMD:

23	22	21	20	19	18	17	16	
								function
0	0	0	0	0	0	0	1	RAM_ID
0	0	0	0	0	0	1	0	RAM_NXPTR
0	0	0	0	0	0	1	1	RAM_OATPTR
0	0	0	0	0	1	0	0	RAM_CURPTR
0	0	0	0	0	1	0	1	RAM_NO
0	0	0	0	0	1	1	0	R_REG
0	0	0	0	0	1	1	1	ADDR_REG
0	0	0	0	1	0	0	0	CHN_REG

23	22	21	20	19	18	17	16	
0	0	0	0	1	0	0	1	RTL_REG
0	0	0	0	1	0	1	0	REQ_REG
0	0	0	0	1	0	1	1	FPID_REG
0	0	0	0	1	1	0	0	TC_REG
0	0	0	0	1	1	0	1	FLAG_REG
0	0	0	0	1	1	1	0	REM_REG
0	0	0	0	1	1	1	1	NO_REG

The Address in the Command Register is only for RAMx accesses (command 1 - 5) valid, address width is 8 bit. Autoincrement will only be supported for RAM Access.

General

- the test access provides read/write access of important internal rams and registers
- test registers are virtuals global registers (SEL signal: pb_vg_tfpi_sel_n / implemented as a daisy chain).
- the single macros are selected by the MID code of the test command register
- CMD specifies/selects one of the macro rams/registers
- the address field is used to access a ram address
- AI: autoincrement : address given in the address field is incremented automatically for each access
- All macros which are not selected by MID drive data output "00000000" and <macro>_TPFI_RDY = '1'. Driving "00000000" would mean not disable the enable line for data out, but to set the output data to "00000000".

Test write access:

1. Write TAC
2. Write TAD

Test read access:

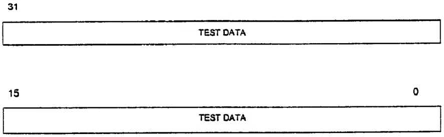
1. Write TAC
2. Read TAD

Typically the macro selected via MID delays the RDY signal until the selected ram/register has been read and the data can be provided at the TFPI interface. No prefetch of testdata is required.

Note: there is no read/write selection in the CMD field; rd/wr's are handled with the TFPI read & write signals

4.3.4 Test Data Register (TD)

Access	: read/write
Address	: $5C_H$
Reset Value	: 00000000_H



TEST DATA:	ram/register test data (read or write)

Note: It is assumed that a TFPI_SEL_N signal is generated for register access of common TAC and TD register. For write / read access the MID (macro ID) is used for addressing one macro: If MID does not match with the hard coded ID, TFPI_D are prepared corresponding the 'daisy chain' requirements.I.e. each macro drives "00000000". These '0' s are ored in the macroshell.

5 Functional Test Specification (Macrolevel)

(use Checklist C06 "Function Checklist - Comparison of Specification vs. Circuit" http://www.hl.siemens.de/lognet/hl_ta/dhb/f.htm)

Appendix F

Title:
Macro Specification DMUR Version 2.2

1 Introduction

1.1 Overview

The Receive Data Management Unit (DMUR) provides direct data transfer from the on-chip Receive Buffer (RB) to the shared memory with minimal host CPU intervention. The on-chip Receive Buffer stores data after protocol (HDLC, Ethernet, ATM,...) processing for each channel, so that the DMUR can initiate a data burst on the system bus (e.g. PCI) and decrease the bus occupancy. A linked list of descriptors associated to each channel is located in the shared RAM and handled by the DMUR. The address generator of the DMUR supports full link list handling. The descriptors can be stored in separate memory location from the data buffers themselves allowing full scatter/gather assembly of packets. In order to have only one read access on the system bus for each descriptor, the current descriptor is hold on-chip. The interrupt vectors generated by the DMUR are transferred to a separate interrupt controller (DMUI). The DMUI will transfer the interrupt vectors of the complete device to the shared memory.

The number of channels is a flexible parameter of the DMUR macro. The DMUR can be used for multichannel devices (M256) as well as for high speed controllers with a reduced number of channels (DSCC4).

1.2 Features

- Data transfer from the central Receive Buffer (RB) to the shared memory
- Support of linked list data structures located in the shared memory and consisting of a descriptor associated to a data section
- Independent channel configuration
- Forward interrupt vectors to the DMUI

1.3 System Integration

The DMUR has four interfaces:

- FPI Initiator Bus
- FPI Target Bus
- RB interface
- DMUI interface

The protocol data is read at the RB interface and written to the FPI initiator bus. The descriptors are accessed through the FPI initiator bus. The interrupt vectors are transferred on the DMUI interface bus. Each channel is configured (base address register, command register,...) via the FPI target bus.

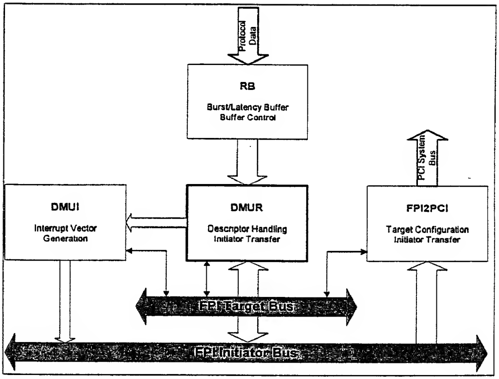


Figure 1
System Integration

1.4 Known Restrictions and Problems

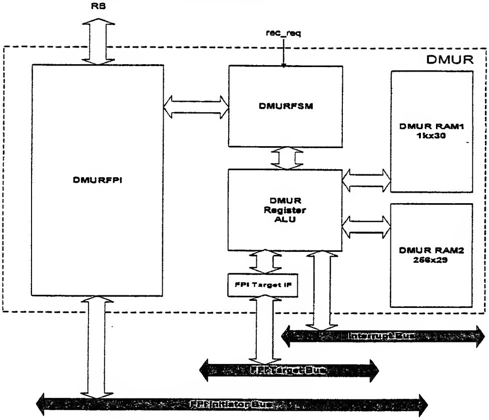
The priorization on the FPI Initiator bus is not an easy task.

DMUR should have priority if new data or a descriptor change is required in the middle of a frame in order to prevent an overflow of the RB (and an abort of the frame). But at the beginning of a frame, the transfer is not time critical.

DMUR and DMUI should have an equal priority on the FPI initiator bus to prevent buffer overflow.

2 Functional Description

2.1 Block Diagram incl. Clocking Regions



The DMUR includes 2 RAMs that store the current descriptor, the next descriptor, the amount of the already transferred data, the amount of left double words in the data section, the status, the mode, the interrupt queue number and the interrupt mask for each channel. The controller unit (DMURFSM) initiates and controlls the data transfer from the receive buffer to the FPI Initiator Bus, the read and the update of the receive descriptors. In order to achieve a maximum data throughput the DMURFPI is directly connected to the receive buffer and the Initiator bus and transfers the data FPI bus compliant, triggerd by the DMURFSM. The ALU and the register set are necessary for burst length and address calculations.

The complete DMUR block and all the interfaces are synchronous to the same CLK signal derived from the FPI initiator bus.

2.2 Normal Operation Description

The DMUR operates with a linked list of descriptors pointing to a data section. For each active channel a linked list is allocated in the external shared memory by the CPU. The descriptor contains the pointer to the next descriptor, the start address of the data section and the reserved size of the data section. The DMUR will update the descriptor with an additional field consisting of the effective number of bytes written in the data section.

For initialization of a channel in the DMUR, the CPU programs the First Receive Descriptor Address (FRDA) register, the interrupt queue, the interrupt mask and the Channel specification Command register with the Receive Init Command (refer to chapter 4.2). These information will be stored at the internal RAM. When the receive buffer requests a data transfer for this channel the DMUR then fetches the descriptor pointed by FRDA from the shared memory, which in turn point to the data buffer and its length as well as the next receive descriptor to go after the reception of the current buffer is completed.

The DMUR gets requests from the Receive Buffer for the transfer of data. The DMUR reads first the request register (located at address 0 on the bus between DMUR and RB). RB returns in the following clock cycle the Request Register, consisting of the channel number (CHN) and the number of words (BL+1) the RB wants to transfer to DMUR and a flag whether the last word contains status information (Data/Status Flag = 1) or not (Data/Status Flag = 0). If the last word in the transfer is a status word, the number of valid bytes in this status word will be provided additionally using the BE bits. In this case the BE bits have the following meaning:

Table 1
Request Register

Bit	31... 27	26	25	24	23.. BLB+16	BLB+15...16	15 .. CNB	CNB-1 0
Function	Reserved	BE 1	BE 0	Data/ Status Flag	Reserved	Burst length BL	Reserved	Channel number (CHN)

BE[0..1]=00: Byte 0-2 are invalid, the word contains only status information

BE[0..1]=10: Byte 0 is valid, Byte 1-2 are invalid user data.

BE[0..1]=01: Byte 0-1 are valid, Byte 2 is invalid user data.

BE[0..1]=11: Byte 0-2 are valid

The DMUR reads the Receive Descriptor corresponding to the channel number CHN and calculates the maximum number of bytes that can be transferred in the current receive data section in the external memory by checking the Byte Number (NO) field.

If NO/4 is greater or equal than the Burst Length (BL+1) value, the DMUR will start a burst transfer with BL+1 words from the Receive Buffer to the FPI2PCI bridge over the

FPI initiator bus. Otherwise, if NO/4 is less or equal than (BL+1), the DMUR will start a Burst Transfer with NO/4 words.

The DMURFSM calculates whether (BL+1) or NO/4 words are transferred and requests the transfer from the DMURFPI. Additionally the start address for the data in the shared memory will be a parameter for the DMURFPI. The DMURFPI reads the protocol data on the RB interface at the data port register (address 1, first DWORD) and stores it internally. The DMUR will then arbitrate the FPI master bus. Once it was granted it will transfer the stored DWORD to the bridge and continue transfering the rest of the burst from the receive buffer directly to the FPI2PCI bridge.

If the data section in the shared memory has been filled with data, the DMUR will update the receive descriptor by writing the complete (C) bit and the number of stored bytes (BNO) in the current data section. An HI interrupt vector will be transferred to the DMUI if the HI bit is set in the receive descriptor. The DMUR will then branch to the next receive descriptor.

In the case where the data section is not full and the frame is not yet ended (with a valid or invalid status), the receive descriptor is not updated in the shared memory (only the on-chip BNO value is updated).

This procedure will continue until the complete Burst Length (available data) is transferred to the shared memory through the FPI initiator bus.

When a frame end is detected and all data have been transferred, the DMUR writes the number of bytes (BNO) stored in the data section in the current descriptor, sets the Frame End (FE) bit and writes the complete (C) bit, no matter if the datasection is full or not. The status of the completed frame will be additionally written into the Receive Descriptor. A FE interrupt vector is transferred to the DMUI, the HI will be set in this vector as well if the corresponding bit is set in the current receive descriptor.

The data transfer is controlled by the HOLD bit, which is located in the Receive Descriptor. If the HOLD bit is not set the current receive descriptor is not the last one in the descriptor chain for this channel and the DMUR will then branch off to the next descriptor.

If the current receive descriptor has its "HOLD" bit set, it is the last one in the descriptor chain. Four different cases have to be distinguished:

a) When the current data buffer has been filled, does not contain the end of a frame (frame based protocolls) and the requested burst length by the RB is not satisfied, the DMUR polls the HOLD bit of the current Receive Descriptor once more. If then the hold bit is removed, the DMUR branches to the next descriptor (NRDP). If the HOLD Bit is still set, an internal Poll Bit will be set and the DMUR does not branch off to the next descriptor for this channel. Additionally an Hold Caused Receive Abort Interrupt (HRAB, HI bit set if necessary) is generated. The status of the descriptor in the shared memory is aborted (RAB bit set), the Complete and the Frame End Bit will be set in the Receive Descriptor. The rest of the frame will be discarded by the DMUR. As long as the HOLD bit remains set, the transferred data from RB for this channel are discarded by the DMUR and for each discarded frame an interrupt with the bits HRAB and RAB set will be generated. No descriptor update will take place.

b) The current data buffer has been filled up and it does contain the end of frame. A Frame End Interrupt (HI bit set if necessary) will be generated, the descriptor will be updated with the FE and C bit and the status of this receive descriptor is error free. With the next request for this channel by the receive buffer, the DMUR repolls the HOLD bit of the current receive descriptor. If then the hold bit is removed, the DMUR branches to the next descriptor (NRDP). If the HOLD Bit is still set, an internal Poll Bit will be set and the DMUR does not branch off to the next descriptor for this channel. As long as the HOLD bit remains set, the transferred data from RB for this channel are discarded by the DMUR and for each discarded frame an interrupt with the bits HRAB and RAB set will be generated. No descriptor update will take place.

c) The current data buffer has been filled up completely (requested transfer length = current NO) and does not contain the end of the frame. The descriptor will be updated immediately (C bit set), an HI Intr will be generated if the HI bit is set. With the next request for this channel by the receive buffer, the DMUR repolls the HOLD bit of the current receive descriptor. If then the hold bit is removed, the DMUR branches to the next descriptor (NRDP). If the HOLD Bit is still set, an internal Poll Bit will be set and the DMUR does not branch off to the next descriptor for this channel. Additionally an Hold Caused Receive Abort Interrupt (HRAB, HI bit set if necessary) is generated. No descriptor update will take place and the data of the current receive buffer request will be discarded. As long as the HOLD bit remains set, the transferred data from RB for this channel are discarded by the DMUR and for each discarded frame an interrupt with the bits HRAB and RAB set will be generated. No descriptor update will take place.

d) The current receive descriptor has its HOLD bit set, but the requested transfer length by the receive buffer is smaller than the reserved data section (NO). All data will be transferred to the reserved data section, no descriptor update will take place neither an interrupt will be generated. This procedure will continue until either case a), case b) or case c) will become true.

When the next receive descriptor is available in the shared memory, the CPU can write the Channel Specification command register (refer to chapter chapter 4.2) with the Receive Hold Reset (RHR) Command, so the internally Poll Bit will be cleared. When the RB requests a new data transfer for this channel, the DMUR still sees the onchip stored HOLD Bit set, but the Poll Bit reseted.This combination forces the DMUR to repoll the HOLD descriptor. The HOLD bit is supposed to be removed, additionally the Next Receive Descriptor Pointer will be read. Then it will branch to this Next Receive Descriptor Pointer. After the RHR command,

- for a frame based protocoll (refer to chapter chapter 4.2.4) the data transferred from RB is discarded until the end of a received frame and the next received frame is related to the next receive descriptor.

– for the transparent mode the data transferred from RB is written immediately to the data section of the next receive descriptor.

If the CPU issues a Receive Hold Reset Command and does not remove the HOLD bit (erroneous programming), no action will take place. Alternatively a Receive Abort or a Receive Off command can be issued by the CPU in order to release a channel from this HOLD state (refer to command description).

If the big endian byte ordering is programmed, the DMUR provides byte ordering swapping for the data transfer between RB and the FPI initiator bus.

Note: The descriptor transfer is always a 32 bit access (no byte swapping).

The following table shows how data is stored in the shared memory depending on the little or big endian byte ordering convention:

Table 2
Little/Big Endian Byte Ordering

BNO	Little Endian				Big Endian			
3	-	Byte 2	Byte 1	Byte 0	Byte 0	Byte 1	Byte 2	-
7	Byte3	Byte 2	Byte 1	Byte 0	Byte 0	Byte 1	Byte 2	Byte3
	-	Byte6	Byte5	Byte 4	Byte 4	Byte5	Byte6	-

2.3 Receive Descriptor Description

The receive descriptor is initialized by the host CPU in the shared memory and is read by the DMUR, when requested so by the host via Receive Init command or after branching from the previous receive descriptor. The receive descriptor is read entirely at the beginning and stored in the on-chip memory. Therefore all information in the next descriptor must be valid when DMUR branches to this descriptor.

FE, C, BNO and status are set by the DMUR, all other values are set by the host.

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Byte Address																
0x00	Rsvd	HOLD	HI	Offset			Rsvd				Descriptor ID					
0x04	Next Receive Descriptor Pointer															
0x08	Receive Data Pointer															
0x0C	FE	C	Rsvd						Status							

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Byte Address																
0x00	NO															
0x04	Next Receive Descriptor Pointer															
0x08	Receive Data Pointer															
0x0C	BNO															

HOLD: Hold

It indicates whether the current descriptor is the last element of a linked list or not:

HOLD=0: A next descriptor is available in the shared memory; after checking the HOLD bit stored in the on-chip memory the DMUR branches to next receive descriptor.

HOLD=1: The current descriptor is the last one for this channel that is available for the DMUR. After completion of the current receive descriptor an HRAB interrupt is generated if the complete frame could not be stored in the data section. The corresponding DMUR channel is deactivated for receive direction as long as the CPU does not request an activation via the Channel Specification command register.

HI: Host Initiated Interrupt

If the HI bit is set, the DMUR generates an HI interrupt vector to the interrupt controller (e. g. DMUI) after transferring all relevant data bytes into the current data section.

NO: Byte Number

This byte number defines the size of the receive data section allocated by the host. The maximum buffer length is 65535 bytes and it has to be a multiple of 4 bytes. The data bytes are stored into the receive data section according to the selected mode (little endian or big endian).

If NO is set to 0, no data will be written to the datasection. The DMUR updates the status of the current receive descriptor with BNO = 0 and sets the Complete Bit. An HI intr will generated, if necessary. Afterwards the DMUR jumps immediately to the next receive descriptor.

Next Receive Descriptor Pointer:

This 32-bit pointer contains the start address of the next receive descriptor. After completion of the current receive descriptor the DMUR branches to the next receive descriptor to continue reception. The receive descriptor is read entirely at the beginning of reception and stored in on-chip memory. Therefore all information in the next descriptor must be valid when the DMUR branches to this descriptor.

This pointer is not used if a receive abort command is detected while the DMUR still writes data to the current receive descriptor. In this case the First Receive Descriptor Address (FRDA) is used as a pointer for the next receive descriptor to be branched to.

Receive Data Pointer:

This 32-bit pointer contains the start address of the receive data section. The start address must be 32-bit (DWORD) aligned.

FE: Frame End

It indicates that the current receive data section (addressed by Receive Data Pointer) contains the end of a frame (frame can be valid or invalid - further information in the status). This bit is set by the DMUR after transferring the last data of the frame from the internal receive buffer into the receive data section in the shared memory. Moreover the BNO is updated, the C bit is set by the DMUR and a Frame End Interrupt is generated.

With the next request for this channel, the DMUR checks the HOLD bit stored in on-chip memory. When HOLD=0 it branches to the next receive descriptor. Otherwise the corresponding DMUR channel is deactivated as long as the CPU does not request an activation via the command register.

C: Complete

This bit is set by the DMUR if

- it completes filling data section normally
- it was aborted by a receiver reset command
- the data sections contains a end of frame (for frame based protocols)

was stored in the receive data section.

BNO: Byte Number of Received Data

The DMUR writes the number of data bytes it has stored in the current data section into BNO field.

Descriptor ID

This field is read by the DMUR and written back in the corresponding interrupt status vector for all channel interrupt vectors. This value provides a link between the interrupt vector and the descriptor, to be used by the software.

Offset

Word (32 bit) offset of unused data section. This field is set by the CPU and is only used by DMUR at the beginning of a frame (will be ignored in transparent mode) and allows the CPU to reserve memory space for an additional header. If the descriptor is the first of a frame the DMUR will write data at the address Receive Data Pointer + Offset, otherwise the field is ignored. The Offset has to be equal or bigger than NO/4. If the Offset is equal to NO/4, the DMUR does not write any data to this datasection, just the complete bit will be set and data will be stored in the data section belonging to the next receive descriptor.

Status

The DMUR writes the status information into the Status byte whenever it sets the C and FE bit field. It is the copy of the status bytes coming from the Receive buffer (refer to chapter 3.2) and has to following coding:

bit 16: RAB

Bit 17: ILEN

Bit 18: CRC

Bit 19: RFOD

Bit 20: MFL

2.4 Interrupts

The DMUR generates 2 kind of interrupts, the command interrupts and interrupts which are related to the transfer mechanism of the DMUR controller for a particualr channel. In order to distinguish these 2 interrupt groups, command interrupts and channel interrupts will be introduced. All command interrupt will be writen to a dedicated queue with a fixed length, channel interrupts can be written to 8 different queues, the designated queue for each channel will assigned during initialization. The interrupt vectors basicly consist of the ID, the channel number and specifific bits.

The Descriptor ID value is written back in the corresponding channel interrupt vector. This provides a link between the interrupt vector and the descriptor, to be used by the software. Additionally the interrupt queue number for this channel will be provided in channel interrupt vectors. The interrupt vectors are transferred to the DMUI and have the following format:

Table 3
Interrupt Vector Format (channel interrupt vector)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
1	0	1	0	1	Q2	Q1	Q0	0	0	Descriptor ID					

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
HI	RAB	FE	HRAB	MFL	RFOD	CRC	ILEN	Channel No							

Command Interrupt Vector

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	CMDC

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	Channel No							

Channel number

Identifies the channel where the interrupt occurred

Descriptor ID

Identifies the receive descriptor, which generated the interrupt

Q2,Q1,Q0: Queuenumber

Identifies the number of the interrupt queue, located in the shared memory

HI: Host Initiated Interrupt

If the HI bit in the receive descriptor is set, this bit will be activated in the interrupt vector when DMUR finishes the current Receive Descriptor (no matter whether a next descriptor is available or not).

FE: Frame Indication Interrupt

FE=1 indicates, that a frame (valid or invalid) has been received completely or was stopped by a Receive Abort, Receive Off command or a HOLD in a receive descriptor. If the frame is valid or not depends on the status bits RAB, HRAB, MFL, RFOD, CRC, ILEN in the vector. As soon as one of these bits is set, errors occured during the reception of the frame. It is also set when the descriptor in which the frame has been finished contained a hold bit (combination Frame End + HOLD).

HRAB: Hold Caused Receive Abort

Issued if the requested amount of data could not be transferred to the shared memory completely because of a HOLD bit set in the current receive descriptor not providing enough memory space for the requested data. An interrupt with RAB and HRAB set will be generated for every complete frame which was lost due to the HOLD bit in the receive descriptor.

RAB

Receive Abort indication. This bit will be set, when the CPU programmed a receive abort or a receive off command and therefore a frame could not be transmitted completely (abort from the CPU side). Additionally it can be set in the status word coming from the receive buffer indicating that the frame was aborted from the serial side. It then will be copied to the interrupt vector as well. It will be also set in combination with HRAB bit for each discarded frame (refer to HRAB description).

The following 4 bits are copied by the DMUR from the status byte coming from RB in the interrupt vector. It includes status information generated by the Protocol Handler.

MFL

Maximum Frame Length. The Packet exceeds the max. allowed frame size.

RFOD

Indicates a Receive Frame overflow. The Protocol handler was unable to transfer data to the RB. As soon RB can store data again, a RFOD status is appended to the frame.

CRC

Cyclic Redundancy Check error

Invalid length (ILEN)

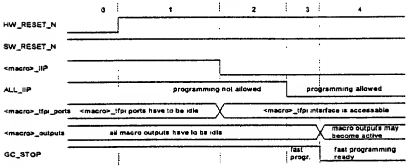
Indicates a frame length not multiple of 8 bits.

CMDC: Command Completed

Identifies a successful channel command performed by the host CPU. The DMUR can accept a new command for this channel.

2.5 Reset Behavior

Initialisation of M256F (macro view)



Note:
HW_RESET_N, SW_RESET_N, <macro>_IIP and GC_STOP have to be implemented in each macro; ALL_IIP is no macro signal

Note:
<macro>_outputs are non TFPI outputs
IIP: Initialisation in progress

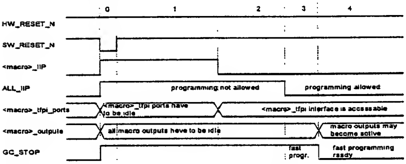
0: HW_RESET_N le active and dsactivates SW_RESET_N; sll macros are reset; GC_STOP set to '1'

1: HW_RESET_N inactive => each macro executee the macro specific initialisation of rams

2: some macros are ready with Initislisation of rams, some not (ALL_IIP Is s or-function of all <macro>_IIPs). The "ready mecros" could be acceseed by TFPI interface, but softwars programming Is not allowed as long as ALL_IIP le active

3: all macros ars rsady with Initialisation => ALL_IIP becomee inactive; software polls ihs ALL_IIP bit after HW_RESET_N; if ALL_IIP = 0 then software programms all macros (in a default "fast programming mode", GC_STOP = 1; reset value of GC_STOP is '1', but can be reset at any time)

4: sfter fsst programming macro outputs (non TFPI) may bacome active and macro has to react on non FPI inputs



0: SW_RESET_N becomes active; all macros (registere) are reset; GC_STOP le active

1: SW_RESET_N inactive => each macro executss the macro specific initialisation of rame

2: some macros ars ready with Initialisation of rame, some not (ALL_IIP Is s or-function of all <macro>_IIPs). The "ready macros" could be accessed by TFPI interface, but software programming Is not allowed as long as ALL_IIP Is active

3: all macroe are ready with initialisation => ALL_IIP becomes inactive, software polle ths ALL_IIP bit sfter SW_RESET_N; if ALL_IIP = 0 then software programms all macros in a "fast programming mode", GC_STOP = 1

4: after fast programming macro outputs (non TFPI) may bacome active and each macro has to react on non FP1 inputs

All internal registers and functions are initialized to known states (RESET state).

In addition to this global reset, each DMA channel can be reseted (turned off) via the command register.

2.6 Functional Test Description

2.7 Production Test Description

3 Macro Interfaces and Signal Description

All signals are active high until otherwise specified. Active low signals are designated by "_N" (FPI mode) appended to their names. To make the design as re-usable as possible, a bus signal whose width is application dependent is specified with one of the following parameters:

Parameter name	Bus Type	Typical value (bits)
		M256F
CNB	Channel Number Bus	8
DBB	Data/Status Bus	32
BLB	Burst Length Bus	6

Note: *Since the maximum burst length is limitied to 64 DWORDS by the PCI2FPI bridge in the M256F the Receive Buffer will never request a data transfer exceeding this limit*

3.1 Signal Description

In the following sections, "Flexible Peripheral Interconnect (FPI) Bus compliant" means that the specified bus uses a subset of the FPI features and satisfies the basic address and data cycle. Not all FPI signals are implemented because default values are sufficient for the application i.e. they can be coded as constants in the hardware. Refer to the FPI bus specification for details of the complete bus.

Two additional out-band signals were introduced at the receive buffer - DMUR interface:

- REC_STAT to indicate if currently transferred word is status instead of pure data. This signal is valid for one cycles in DMA transfer.
- REC_REQ_N to indicate that the Receive Buffer Action Queue (RBAQ) in the RB is not empty.

Table 4
Macro Interfaces and Signal Description

Symbol name	I/O	Function
Control Signals		
sysclk	I	system clock
reset_n	I	Hardware reset
scanmode	I	Scanmode
gc_stop	I	global stop signal, used for fast RAM programming by the CPU
dr_iip	O	Initialisation in progress, when active the DMUR initialize its internal RAMs
Receive Buffer (RB) interface		
dr_rec_a	O	1 bit address bus 0 => status port (request register). 1 => data port.
dr_rec_rd_n	O	Read Control Output
rec_d[DBB-1:0]	I	Output data/status from RB to DMUR controller
rec_rdy	I	End of data transfer indication. 0 => DMUR should insert wait state. 1 => RB will finish transfer during this clock cycle.
rec_stat	I	Current transferred word is status
rec_req_n	I	Service request from RB to DMUR controller. Asserted when RBAQ is not empty.
Interrupt Controller (DMUI) Interface		
dr_int_req_n	O	Interrupt Bus Request Line
int_gnt_n	I	Interrupt Grant Line (from interrupt bus arbiter)
dr_int_d[DBB-1:0]	O	Interrupt Vector

Table 4
Macro Interfaces and Signal Description (cont'd)

Symbol name	I/O	Function
dr_int_d_en[7:0]	O	Output Enable Bus

FPI Target Interface

tfpi_a[8:2]	I	Address bus.
tfpi_d[DBB-1:0]	I	Data bus input
tfpi_rdy	I	Ready Input, other target will finish data cyle in the clock cycle (tfpi_rdy = 1) or it will insert Waitstates

tfpi_wr_n
tfpi_rd_n
	I
	I
		Read/Write controls. Following codes are defined:
		WR_N = 0; RD_N = 1 => data written to DMUR
		WR_N = 1; RD_N = 0 => data read from DMUR

tfpi_vc_sel_n	I	DMUR Slave select (registers of virtuell channel specification). 0 => Address is valid 1 => Address is not valid
tfpi_vg_sel_n	I	DMUR Slave select (global registers). 0 => Address is valid 1 => Address is not valid
dr_tfpi_d[DBB-1:0]	O	Data bus output
dr_tfpi_d_en[7:0]	O	Output Enable Bus for data bus (one enable line for 4 data lines)
dr_tfpi_rdy	O	End of transfer indicator: 0 => Master should insert wait states 1 => DMUR will complete transfer in this cycle
dr_tfpi_rdy_en	O	Output Enable signal for Ready Output

FPI Initiator Bus

ifpi_gnt_n	I	DMUR INITIATOR Grant Line
ifpi_opc[3:0]	I	Operation Code (Input for split read response)
ifpi_d[DBB-1:0]	I	Data Bus (for split block read)
ifpi_sel_n	I	Select Signal for DMUR (for split block response)
ifpi_ack[1:0]	I	Slave Response code

Table 4
Macro Interfaces and Signal Description (cont'd)

Symbol name	I/O	Function
ifpi_rdy	I	Slave on data bus will be able to finish transaction in current clock cycle
dr_ifpi_req_n	O	DMUR INITIATOR Bus Request Line
dr_ifpi_rd_n	O	Read Control
dr_ifpi_rd_n_en	O	Read Control Output Enable
dr_ifpi_wr_n	O	Write Control
dr_ifpi_wr_n_en	O	Write Control Output Enable
dr_ifpi_abort_n	O	Abort Signal
dr_ifpi_abort_n_en	O	Abort Output Enable
dr_ifpi_opc[3:0]	O	Operation Code
dr_ifpi_opc_en	O	Operation Code Output Enable
dr_ifpi_tc[BLB-1:0]	O	Transfer Count (max. burst size 64dwords)
dr_ifpi_tc_en[1:0]	O	Transfer Count Output Enable (one enable line drives 4 data lines)
dr_ifpi_a[DBB-1:2]	O	Address Bus
dr_ifpi_a_en[7:0]	O	Address Bus Output Enable Bus
dr_ifpi_d[DBB-1:0]	O	Data Bus
dr_ifpi_d_en[7:0]	O	Data Bus Output Enable Bus
dr_ifpi_tag[3:0]	O	Tag Bus (used to transfer the master ID for Split Block Transfers)
dr_ifpi_tag_en	O	Tag Bus Output Enable
dr_ifpi_rdy	O	DMUR will be able to finish transaction in current clock cycle
dr_ifpi_rdy_en	O	Ready Output Enable

(*) These signals connect DMUI and the arbitration logic on the DMUI bus.

3.2 Data Flow and Functional Timing

3.2.1 FPI Initiator Bus

Refer to the SIEMENS FPI Bus for Munich-Macros Specification.

3.2.2 FPI Target Bus

Refer to the SIEMENS FPI Bus for Munich-Macros Specification.

3.2.3 RB Interface

As soon as the receive buffer wants to transfer data to the system memory through the DMUR it asserts its rec_req_n signal. If the DMUR is currently in idle state, it initiates an address cycle by asserting the read signal with address 0. During the data cycle the RB returns the RB-DMUR request register and asserts rec_rdy_n (one wait state will be typically inserted). Based on the requested burst and the channel number, the DMUR set up the transfer by assigning the destination address and the burst length. By asserting the read signal and the address (address 1 data buffer) the DMUR transfers BL+1 words with either individual or overlapped cycles. In best case, a burst cycle can occur without wait states but internally, the PMR and configuration interfaces have higher priority.

Figure 2 illustrates a basic transfer (the RB inserts typically 1 wait state at the beginning of the data transfer, 2 wait states in cycles 10 and 11 are inserted due to collision with the protocoll machine interface). Note that the DMUR needs some cycles to initiate the transfer after reading the request register for loading the channel context and address calculation.

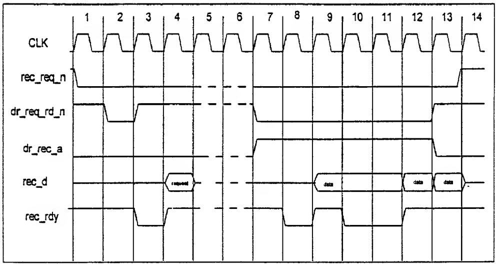


Figure 2
Basic transfer between DMUR and RB

The efficiency of the transfers improves with burst length. The peak throughput (not sustainable) can approach 1 word/clock with long bursts. However it is ultimately limited by PMR. Each word the PMR transfers during a RB-DMUR data cycle insert 2 wait states in RB_RDY_N.

Note that DMUR must always read the request register before a data transfer. The Burst Length field in RBAQ is actualized after each data access and the entry in RBAQ is removed after all the corresponding data have been transferred to the DMUR.

If the burst length ((BL+1) *4) in the request register of the RB is bigger than NO of the current datasection, the DMUR will split the transfer. It will first fill the current datasection, update the status of the current receive descriptor, generate an HI interrupt (if necessary) and branch to the Next Receive Register pointed by the Next Receive Descriptor Pointer. Then it will continue transferring data.

The side band signal rec_stat will be asserted when the current transferred data word at the interface RB-DMUR contains status information (i.e. Frame End).

Figure 3 shows a data transfer to 2 data sections (2 different receive descriptors) and the use of the sideband signal rec_stat (the typical wait state at the beginning of a transfer is not shown in the figure)

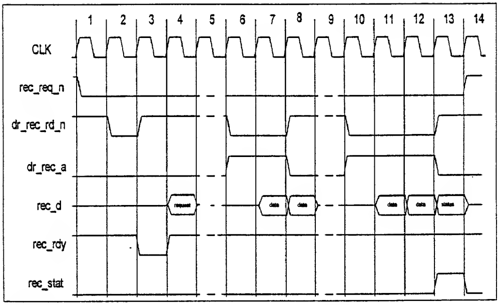


Figure 3
Data Transfer to 2 different Receive Descriptors of 1 channel

In case of a status the word has the following format:

Table 5
Status word

Bit	31	30	29	28	27	26	25	24	23...16	15:8	7:0
Function	FE	MFL	RFOD	CRC	ILEN	RAB	rsvd	rsvd	Byte 2	Byte 1	Byte 0

The BE Bits in the request register of the RB indicate the valid bytes in this statusword. RAB (Receive Abort), ILEN (Invalid Length), CRC (CRC error), RFOD (Receive Frame Overflow), MFL (Maximum Frame Length) and FE (Frame End) are status bits generated by the protocol handler and are mapped in the interrupt vector (refer to chapter chapter 2.4). In case of external or internal problems, indicated by the bits RAB, ILEN, CRC, RFOD, MFL, the FE bit (within the status word) will not be set (exception Receive OFF indication FE and RAB set). When the DMUR updates the current receive descriptor with the error status it then sets the FE bit. New data for this channel, indicated by a new request, are related to a new frame.

3.2.4 DMUI Interface

The DMUR transfers the interrupt vectors to the DMUI (Interrupt controller) which will arbitrate the FPI initiator bus and transfer the vector in the corresponding interrupt queue in the shared memory. The interface between DMUR and DMUI is a bus shared by all the blocks generating interrupts. Therefore the transfer starts with an arbitration cycle. Once the bus is granted to DMUR, it deactivates its request line and writes the vector to the interrupt vector data bus in the next cycle (because no address cycle is needed). The interrupt controller will not insert wait states. Since more than one block is working on the interrupt bus, it might take some cycles until the request is granted. The minimum latency will be one clock.

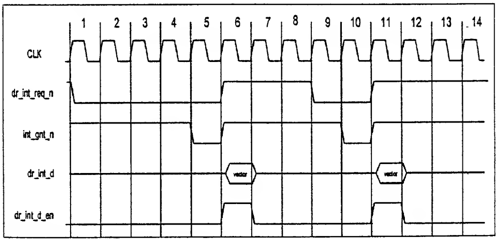


Figure 4
Interrupt Vector Transfer to the DMUI

3.3 Macro Functional Test

3.4 Macro Production Test

4 Register Description

4.1 Register Overview

The DMUR macro will be used in a system. Within the system it will happen that certain configuration information will be used by more than one macro. System programming would be very ineffective if these configuration information will be stored at different locations in different macros. That's why a broadcast programming will be introduced. Channel configuration information will be stored in a set of registers, selected by the signal tfpi_vc_sel_n. Another set of registers, selected by the signal tfpi_vg_sel_n, contains global information for all macros and testmode information. Not all channel and general information are relevant for the DMUR macro, every macro implements only the registers which contain necessary information.

All registers will be accessed via the target FPI bus, inside the macro the SMIF-BPI interface will control this accesses.

Essential for all slave register accesses are fast write accesses (i.e. no PCI wait states) and that a mechanism is provided to ensure that accesses are completed internally before the next command is given (i.e. no erroneous overwrite of data).

Read accesses are non critical, but a debug read back of all registers (virtual and global) must be ensured;

Test mode accesses are non critical and can deliver deliberate results; i.e. read back RAM contents

The following chapter describes a subset of registers, which are necessary for programming the DMUR macro. The reset values are the driven values from the DMUR, in the system environment they might be different (non implemented bits by the DMUR might vary).

4.2 Detailed Register Description of the Virtual Channel Specification (selected by tfpi_vc_sel_n)

4.2.1 Virtuell Channel Specification Command Register (CSPEC_CMD)

Access	: read/write
Address	: 00_H

Reset Value : 00000000_H

31 24	23 16
rsvd	CMD_REC(7:0)

15	0
00	CHAN(7:0)

Note: The reserved bits are implemented by other macros within the system

The following commands will be supported:

Receive Init

This command will cause the initialization of the particular channel CHAN(7:0). The DMUR will store the First Receive Descriptor Address (Register CSPEC_FRDA) in its on-chip memory. When this task is completed a Command Completed Interrupt CMDC vector is transferred to DMUI. Afterwards the DMUR will wait for the first request for this channel from the Receive Buffer. Based on this request the DMUR will fetch the entire receive descriptor pointed by the CSPEC_FRDA register.

Receive Abort

If a receive abort is detected the current receive descriptor will be suspended. The DMUR stores the FRDA in its on chip RAM and generates a CMDC Interrupt.

In transparent mode, an Receive Abort Interrupt will be generated (RAB and HI if necessary), the C bit will be set in the old receive descriptor and the RAB bit is set to one in the status. Then the DMUR jumps immediately to the Receive Descriptor pointed by FRDA triggered by the next request from the receive buffer for this channel. The data from this request will be sent to the data section of this new receive descriptor.

For frame based protocolls (HDLC,PPP) two scenarios are possible. With the next request for this channel, the DMUR recognizes the Abort Command. When the DMUR has finished the previous transmission for this channel with an end of frame (receive descriptor already updated, interrupt generated), new data for this channel are related to the new receive descriptor pointed by FRDA. If the last transferred data for this channel did not contain an frame end, the rest of the received frame, which was only transferred partially, will be discarded. The C and FE bit will be set in this receive descriptor, the status contains the activated RAB bit. The same information will be sent in an Interrupt vector (RAB, FE and HI - if necessary, set). With the next frame the DMUR will branch to the next receive descriptor pointed by the FRDA.

If the data section of the last receive descriptor was filled completely (does not contain a end of frame), the complete bit will be set in the status of this descriptor. The CPU issues a Receive Abort Command before the next request for this channel will appear from the receive buffer. In this case, the DMUR retrieves the receive descriptor pointed by FRDA. This descriptor will be used in order to inform the CPU about the aborted frame with BNO=0 (no data will be written to the data section), the C Bit, the RAB in the status and the FE (only for HLDC). Additionally the corresponding interrupt will be generated. The datsection of the next receive descriptor (pointed by next receive descriptor pointer) will be used in order to store the data of the current request (transparent mode) or the data of the next new frame.

This command also can be used in order to release a channel from the HOLD state. The HOLD descriptor will not be touched anymore since the update already took place, the HOLD bit will not be repolled. No interrupt will ge generated (besides the CMDC), in TMA mode the DMUR jumps immediately to the new receive descriptor pointed by FRDA, in frame based protocolls data of a new frame are related to the receive descriptor pointed by FRDA.

Receive Off

This command will not be durectly interpreted by the DMUR. As long as the DMUR does not see the Receive OFF status word (FE+ Abort bit set; if end of frame and receive off occur at the same time in the protocoll machine, the receive off command will be given priority and so only one receive off status word (FE + Abort set) will be forwarded to the DMUR), it works normally.

If this status word is seen by the DMUR and a frame has not been transferred completely and the current receive descriptor has not been updated yet, it will generate a Receive Abort Interrupt (FE, RAB and HI, if necessary, set). The current receive descriptor will be released setting the C, FE and write the abort status. Afterwards a Command Complete interrupt will be generated.

If the Receive Off statusword is seen by the DMUR and a frame has not been transferred completely and the current receive descriptor already has been updated, the DMUR retrieves the next receive descriptor, writes BNO with 0 and sets FE, C and RAB in the status. Additonally the corresponding Receive Abort Interrupt (FE, RAB and HI, if necessary) will be generated, afterwards the CMDC Interrupt.

If this receive off statusword (FE and Abort set) is seen by the DMUR and the frame was transferred completely, just the CMDC interrupt will be generated.

The channel is turned off. The next command for this channel must be an Receive Init. The DMUR does not expect any data for the channel after the receive off status word was read by the macro until the next initialization.

The Receive Off command also can be used to release a channel, which is on HOLD. No matter if a frame was transferred completely or not, just a CMDC Interrupt will be generated.

Receive Debug

All the registers (besides CSPEC_MODE_REC, since PMD value will be implemented and used by various macros - other macro drives this value during read) of the virtual channel specification will be read with this command.

Receive Hold Reset

Always when the CPU removes the HOLD bit in the receive descriptor chain of the channel additionally it has to write the Receive Hold Reset Command. Internal action refer to description of the polling mechanism (chapter 2.2).

Issuing the Receive Hold Reset Command bit will be accepted immediately, but no acknowledgement (CMDC INTR) will be performed. After executing one of the other commands (Init, Abort, Off) an Command Completed Interrupt (CMDC) will be generated. The CPU may not write another command for this channel into the command register before the CMDC INTR occured. The commands are coded in the following way:

Command Table Receive:

23	22	21	20	19	18	17	16	
Rsvd	Rsvd	Rsvd	Debug	HOLD RESET	ABORT	OFF	INIT	function
0	0	0	0	0	0	0	1	receive init
0	0	0	0	0	0	1	0	receive off
0	0	0	0	0	1	0	0	receive abort
0	0	0	0	1	0	0	0	receive hold reset
0	0	0	1	0	0	0	0	receive debug
0	0	0	0	0	0	0	0	receive nop

With the writing of Virtual Channel Specification Command Register the contents of all registers belonging to Virtual Channel Specification Broadcast programming is valid.

4.2.2 Virtuell Channel Specification Mode Receive Register (CSPEC_MODE_REC)

Access : write

Address : 04_H

Reset Value : 00000000_H

31					
0	0	0	rsvd	rsvd	rsvd

15														0
0	rsvd	rsvd	rsvd	rsvd	rsvd	rsvd	rsvd	0	0	0	0	0	0	PMD(1:0)

Note: The reserved bits are implemented by other macros within the system

PMD(1:0): Protocoll Machine Mode

For each channel, an information is needed whether the protocol is frame based (HDLC, PPP, Ethernet,...) or transparent. In the transparent mode no status byte containing FE is received from RB.

4.2.3 Virtual Channel Specification Buffer (CSPEC_BUFFER)

Access : read/write
Address : 20_H
Reset Value : 00000000_H

31	
rsvd	rsvd

15				0
rsvd	rsvd	0	RQUEUE(2:0)	rsvd

RQUEUE: The generated receive interrupts will be sent to this interrupt queue

Note: The reserved bits are implemented by other macros within the system.

4.2.4 Virtual Channel Specification FRDA (CSPEC_FRDA)

Access : read/write

Address : 24_H

Reset Value : 00000000_H

31	0
FRDA(31:0)	

FRDA(31:0): First Receive Descriptor Address (dword aligned)

4.2.5 (Virtual) Channel Specification Interrupt Mask

Access : read/write

Address : $2C_H$

Reset Value : 00000000_H

31							Tran smit								
0	rsvd	0	rsvd	0	0	0	0	rsvd	rsvd	0	0	0	0	0	rsvd

15							Rec eive								0
0	RAB	RFE	HRAB	MFL	ROFD	CRC	ILEN	rsvd	rsvd	rsvd	0	rsvd	SD	0	RCC

Interrupt Generation Mask: These bits correspond to the respective channel and command interrupts, if set to '1', the corresponding interrupt will not be generated by the device

Note: The reserved bits are implemented by other macros within the system

4.3 Detailed Register Description of the Global registers (selected by tfpi_vg_sel_n)

4.3.1 Configuration Register 1 (CONF1)

Access : read/write

Offset Address : 40_H
Reset Value : 00000000_H

31GC						GC	GC				
rsvd	0	0	0	0	0	rsvd	rsvd	0	0	rsvd	rsvd

15								
rsvd	rsvd	rsvd	rsvd	rsvd	rsvd	rsvd	LBE	rsvd

LBE: Little Endian/Big Endian Byte Swap; Big Endian: LBE = 1

Note: The reserved bits are implemented by other macros within the system

4.3.2 Test Command Register (TAC)

Access : write/read
Offset Address : 58_H
Reset Value : 00000000_H

31					
MID	0	0	0	AI	CMD

15			0
0	0	0	ADDRESS

MID:	Macro ID Code (0101 for DMUR)
AI:	Auto Increment Function
Address:	internal address
CMD:	Command (select of ram and register)

CMD:

23	22	21	20	19	18	17	16	
								function
0	0	0	0	0	0	0	1	RAM1_NO
0	0	0	0	0	0	1	0	RAM1_NDPTR
0	0	0	0	0	0	1	1	RAM1_RDPTR
0	0	0	0	0	1	0	0	RAM1_CDPTR
0	0	0	0	0	1	0	1	RAM2
0	0	0	0	0	1	1	0	R1_REG
0	0	0	0	0	1	1	1	R2_REG
0	0	0	0	1	0	0	0	ADDR_REG
0	0	0	0	1	0	0	1	IV_REG
0	0	0	0	1	0	1	0	NO_REG
0	0	0	0	1	0	1	1	BNO_REG
0	0	0	0	1	1	0	0	STRA_REG
0	0	0	0	1	1	0	1	TNOBL_REG
0	0	0	0	1	1	1	0	FPID_REG
0	0	0	0	1	1	1	1	FPIA_REG

Macro ID Code: Bit 31:28 = '0101' for DMUR

The Address in the Command Register is only for RAMx accesses (command 1 - 5) valid, address width is 8 bit. Autoincrement will only be supported for RAM Accesses.

General

- the test access provides read/write access of important internal rams and registers
- test registers are virtuals global registers (SEL signal: pb_vg_tfpi_sel_n / implemented as a daisy chain).
- the single macros are selected by the MID code of the test command register
- the test access provides read/write access of important internal rams and registers
- CMD specifies/selects one of the macro rams/registers
- the address field is used to access a ram address

- AI: autoincrement : address given in the address field is incremented automatically for each access
- All macros which are not selected by MID drive data output "00000000" and <macro>_TPFI_RDY = '1'. Driving "00000000" would mean not disable the enable line for data out, but to set the output data to "00000000".

Test write access:

1. Write TAC
2. Write TAD

Test read access:

1. Write TAC
2. Read TAD

Typically the macro selected via MID delays the RDY signal until the selected ram/register has been read and the data can be provided at the TFPI interface. No prefetch of testdata is required.

Note: CMD/Address have to be defined for each macro; there is no read/write selection in the CMD field; rd/wr's are handled with the TFPI read & write signals

4.3.3 Test Data Register (TD)

Access	: read/write
Address	: $5C_H$
Reset Value	: 00000000_H

31
TEST DATA

15	0
TEST DATA	

TEST DATA: ram/register test data (read or write)

Appendix G

Title:
Macro Specification
TB Version 2.1

1 Introduction

1.1 Overview

The Transmit Buffer (TB) provides channelwise buffering of raw data/status words between a Data Management Unit Transmit (DMUT) and a Protocol Machine Transmit (PMT). This data is stored in form of (internal) linked lists for all logical channels. These linked lists are pre-allocated according to bandwidth requirements of the respective channels. In order to avoid transmit underrun conditions each channel buffer has two control parameters for smoothing the filling/emptying process (Transmit Threshold, Request Threshold)

The transmit enable threshold value has following impact:

- Transmission of channel data is started if more than the transmit threshold number of words are he stored in the internal transmit buffer or if at least one word with a complete indication (CI bit, see chapter ...) is stored in channel buffer. Otherwise an empty indication (TB_PTEMPTY) will be delivered to PMT. This feature is always active. Typically after a configuration command or after 'TB empty' condition, but also while frame oriented data transfer this feature reduces the probability of data underrun.

The burst threshold value has following impact:

- As soon as the amount of empty channel buffer locations gets above the threshold value the TB will request data from the transmit DMU (DMUT) .

TB works mode and frame independent. TB transfers data (even the status bit) absolutely transparent and fulfills a channelwise buffering and an efficient data burst request generating.

1.2 Features incl. performance, number of gates

- Number of gates: depends strongly on TB dimensions
- Performance: The overall limitation of data throughput is the PMT interface: each 5th cyle a read access to TB is allowed.
- SMIF register interface or alternatively Target FPI interface for configuration of TB (transmit init and transmit off commands), system test (read of programmable parameter 'burst threshold', 'transmit threshold' and 'channel buffer size' via channel debug command) and testmode (read/write of all internal rams via indirect test register access)
- Interface to DMUT and PMT is FPI like
- interrupt controller (IC) interface with channelwise interrupt generation (configuration fail interrupt, not maskable)
- Programmable FIFO size for each channel (ITBS)
- Programmable burst threshold for DMUT request generation (BTC) and transmit enable threshold (TTC)

1.3 System IntegrationSystem Integration and Application

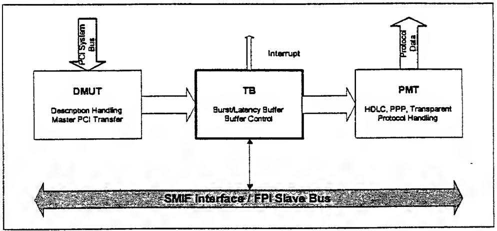


Figure 2
System Integration

The TB has four interfaces:

- SMIF / FPI Slave Interface for programming and system test
- DMUT interface for burst capable data transfer from external memory
- PMT interface for protocoll interface
- DMUI interrupt interface

1.4 Known Restrictions and Problems

The maximum data throughput depends on data bus width, system clock frequency and the programmed configuration of TB.

PMT limitation: The maximum speed of PMT interface is 1 data transfer per 5 cycles. This is a short term maximum througput. TB is able to handle an average througput of 1 data per 20 clock cycles.

With higher burst thresholds and less number of activated channels this average increases.

A further limitation is the behaviour of TB-DMUT-interface: DMUT allows no interleaving of burst request and data transfer, i.e. first a read of request register is done, then either the data are served or the request is stored in DMUT (e.g. HOLD condition in DMUT, see DMUT specification). Afterwards the read of request register is done.

2 Functional and Test Description

2.1 Block Diagram incl. Clocking Regions

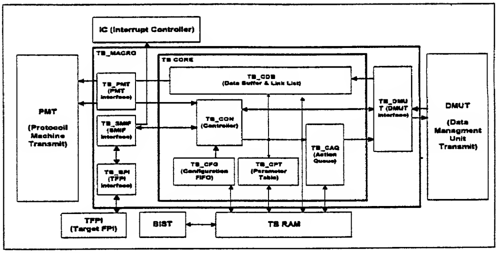


Figure 3
TB Block Diagram

As described in the **Figure 3**, the Transmit Buffer consists of

- Central Data Buffer and link list (TB_CDB)
- Central Parameter Table (TB_CPT) and configuration FIFO (TB_CFG)
- Central Action Queue (TB_CAQ)
- Controller (TB_CON).

Those blocks build the core functionality. Aditionally TB has a FPI like DMUT (TB_DMUT) and a FPI like PMT (TB_PMT) interface. TB macro has an internal SMIF interface (TB_SMIF). Therefore TB offers both

- SMIF and
- TFPI interface (TB_BPI implementation is an optional part of TB Macro).

2.2 Normal Operation Description

After reset all rams are initialized: each ram address is written with idle values and a global link list is set. If initialisation is ready, TB asserts TB_IIP inactive.

Afterwards TB could be configured via TFPI/SMIF interface: a channel command consists of 2 write accesses (TB point of view): first the channel specification buffer register (CSPEC_BUFFER) has to be written with the buffer parameters, then a write of the channel specification command register (CSPEC_CMD) specifies the channel number and the desired command (init, off).

All commands are written into a configuration fifo. This fifo can't be read before GC_STOP is deasserted. Therfore TB won't generate a data request and can't transfer data to PMT during this 'stop' phase.

When GC_STOP is inactiv, the configuration fifo is read.

A transmit init command leads to an allocation of the requested buffer size (ITBS) and writes the according channel number into a action queue. When this action queue is read a data request to DMUT is asserted. After an init command, TB always requests a number of ITBS data from DMUT. Later, the number of requested data will be a function of BTC.

A transmit off command deactivates the channel and deletes all stored channel data. The previous allocated buffer locations become available. The next request to DMUT indicates with DEL=1, that this channel is switched off and all stored requests in DMUT have to be cancelled. RTL has to be ignored and no further requests are generated.

As long as no or less then TTC data are stored in TB, each PMT request is answered with an TB_EMPTY indication.

Incoming data from DMUT will be stored in TB data buffer (TB_CDB) by using the first element from a free pool list. A read request from the PMUT causes the TB to read next data in the channel data buffer (link list) and to add the read location to the free pool list.

Table 1
Request Register for data request from TB to DMUT

Bit	P1			P2	P3			P4	P5
Function	DEL	Reserved		RTL (requested transfer length)		Reserved		CHN (Channel number)	

Table 2
Request Register for data transfer acknowledge from DMUT to TB

P1: tb_dt_crdel_pb_c
P2: tb_dt_crbl_lb_c
P3:tb_dt_crbl_rb_c

Bit	P6			P2	P3			P4	P5
Function	CI	Reserved		STL (served transfer length)		Reserved		CHN (Channel number)	

P4:tb_dt_crchn_lb_c
P5:tb_dt_crchn_rb_c
P6:tb_dt_crci_pb_c

The status of each logical channel is stored in the Parameter Table TB_CPT. As soon as the amount of data stored in the TB gets below the programmable threshold (buffer size minus burst size) for a specific channel, an entry to the action queue (TB_CAQ) is made. TB_CAQ is a FIFO which is dimensioned according to the number of channels (depth = number fo channels). For each channel only one entry is possible. Only the channel number is stored. When the next entry of TB_CAQ is read, a request TB_DTREQ from TB to DMUT is genearated with the number of actual empty buffer locations. Both channel number (CHN) and the number of words (RTL) are written to the request register (read access, refer to **Table 1**). After TB_DTREQ the DMUT has to read the request register and then read the corresponding number of data via external bus interface (PCI) and write them back to TB: before data transfer DMUT writes channel number (CHN) and the number of served length (STL) into request register (write access, refer to **Table 2**). If the last word of the data transfer will be a word that allows data transfer to PMT independent of number of data stored in TB, CI is activated. In some application a frame end (FE) or abort (TAB) indication could force to set CI. CI=1 enables transmission of all data stored in TB.

Unserved or uncompletely served TB requests are stored in DMUT. In HOLD condition TB will continue to request data from DMUT if the request condition is valid. The unserved requests are stored in DMUT. After transfer of the remaining data to PMT TB will become empty. Afterwards TB sets the TB_EMPTY line to PMT. PMT will decide if it is an abort condition or not (in the first case generate an underrun interrupt and in the second case send the idle character). This channel remains inactive as long as the host CPU does not request an activation of DMUT. After activation of DMUT all stored requests have to be served to TB. In case of no activation a 'transmit off' switches this channel to 'idle' condition and gives all reserved buffer locations free.

Status bits and flags have no impact to TB behaviour. E.g. FE (frame end), TAB (transmit abort) and BE (byte enabales) bits are transferred transparently from DMUT through TB to PMT as all other data.

Transfers to the PMT are performed with single words for each channel. The TB provides a control flag (TB_PTSTAT) indicating the word type.

2.2.1 TB Parameter Table (TB_CPT)

The TB_CPT RAM provides a control word per channel for buffer management:

- Channel burst and transmit enable threshold, buffer size
- Count of empty words in channel's buffer chain not already sent to DMUT
- Count of stored words in channel's buffer chain
- Pointers to start and end of buffer chain (CDB read and write addresses)
- Complete channel buffer status and complete indication pointer

RAM size:

MaxNumChan (MNC) words X (6N + BTB + TTB + 4) bit RAM

Table 3: All channel parameters:

ITBS	BTC	TTC	FIRST	AQE	WAIT	CIPV	CIP	FCTR	ECTR	WP	RP
(N)	(BTB)	(TTB)	(1)	(1)	(1)	(1)	(N)	(N)	(N)	(N)	(N)

ITBS: individual transmit buffer size

total buffer size reserved for channel. Max size is $(2^{(N)}-1)$ entries, min. size is 1 entry. Summary of all buffers is DBS entries.

BTC/TTC: burst threshold code/tranmsit enable code

FIRST: wait for first data after configuration

The first data entry after configuration has special conditions

AQE: action queue entry

action queue entry (only one entry per channel allowed);

WAIT: transmission wait until transmit enabled

Channel start of transmit disabled until threshold reached or one complete frame is stored in channel buffer.

CIPV: pointer to last data with a complete indication flag (CI) is valid.

CIP: pointer to last data with a complete indication flag (CI).

FCTR: full counter

number of filled locations in channel buffer. If FCTR = 0 an TB_EMPTY indication is set.

ECTR: empty counter

number of empty locations in channel buffer. If ECTR is bigger than threshold, an action queue entry has to be made

RP: read pointer

Pointer for PMT data requests; RP shows next location to be read.

WP: write pointer

Pointer for DMUT data transfers; WP shows to last written location.

TB_CPT block also includes a free pool pointer (**FPP**) and a free pool counter (**FPC**).

note: N is dependent on number of words DBS in the CDB RAM.
$2^{N-1} < DBS <= 2^{N}$

2.2.2 TB Data Buffer and link list (TB_CDB)

The TB_CDB size (DBS) is influenced by:

- number of channels supported
- sum of channel bit rates
- thresholds
- maximum bus latency

The data buffer locations are connected to a logical buffer by a pointer link list. The link list has the width N and the depth DBS. Each entry specifies the next data buffer element.

2.2.3 TB Action Queue (TB_CAQ)

FIFO to buffer requests from TB to the DMU Transmit. An entry specifies only the channel number:

Size: number of channels x channel bus width

2.2.4 TB Configuration (TB_CFG)

FIFO to buffer the configuration requests from TFPI/SMIF,

Table 4: All locations:

ITBS (N)	BTC (BTB)	TTC (TTB)	CHN (CHN)

A buffer create command forces an entry with ITBS>0, a buffer delete command forces an entry with ITBS=0.

Size: number of channels x (N + BTB + TTB + CHN)

ITBS: individual transmit buffer size

total buffer size reserved for channel. Max size is $(2^{(N)}-1)$ entries, min. size is 1 entry. Summary of all buffers is DBS entries.

BTC/TTC: burst threshold code/transmit enable code

CHN: channel number

2.3 Reset Behavior

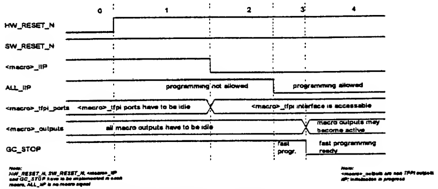
The TB is reset with the line RESET_N.

All internal registers are reset while reset condition; after reset all rams are initialized to known states (RESET state), while TB_IIP is active. The initialisation time depends on the complete data buffer size, because an initial link list in the link list ram has be to generated.

After initialisation TB_IIP becomes inactiv and transmit buffer TB is ready for configuration and data transfer.

After reset/initialisation all interfaces (PMT, DMUT, IC) are inactive and the registers are accessible via the TFPI bus / SMIF for configuration. For a fast configuration the line GCSTOP can be activated to suppress all other interface communication.

Initialisation of Toplevel (macro view)


0
1
2
3
4
HW_RESET_N
SW_RESET_N
<macro>_IIP
ALL_IIP
programming not allowed
programming allowed
<macro>_tfpi_ports
<macro>_tfpi ports have to be idle
<macro>_tfpi interface is accessable
<macro>_outputs
all macro outputs have to be idle
macro outputs may become active
GC_STOP
fast progr.
fast programming ready
Note:
HW_RESET_N, SW_RESET_N, <macro>_IIP and GC_STOP have to be implemented in each macro, ALL_IIP is no macro signal
Note:
<macro>_outputs are non TFPI outputs
IIP: initialisation in progress

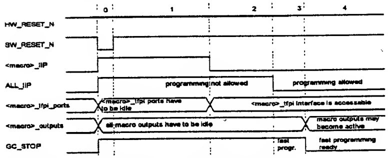
0: HW_RESET_N is active and deactivates SW_RESET_N; all macros are reset; GC_STOP set to '1'

1: HW_RESET_N inactive => each macro executes the macro specific initialisation of rams

2: some macros are ready with initialisation of rams, some not (ALL_IIP is a or-function of all <macro>_IIPs). The "ready macros" could be accessed by TFPI interface, but software programming is not allowed as long as ALL_IIP is active

3: all macros are ready with initialisation => ALL_IIP becomes inactive; software polls the ALL_IIP bit after HW_RESET_N; if ALL_IIP = 0 then software programms all macros (in a default "fast programming mode", GC_STOP = 1, reset value of GC_STOP is '1', but can be reset at any time)

4: after fast programming macro outputs (non TFPI) may become active and macro has to react on non FPI inputs



0: SW_RESET_N becomes active; all macros (registers) are reset; GC_STOP is active

1: SW_RESET_N inactive => each macro executes the macro specific initialisation of rams

2: some macros are ready with initialisation of rams, some not (ALL_IIP is a or-function of all <macro>_IIPs). The "ready macros" could be accessed by TFPI interface, but software programming is not allowed as long as ALL_IIP is active

3: all macros are ready with initialisation => ALL_IIP becomes inactive; software polls the ALL_IIP bit after SW_RESET_N; if ALL_IIP = 0 then software programms all macros in a "fast programming mode", GC_STOP = 1

4: after fast programming macro outputs (non TFPI) may become active and each macro has to react on non FPI inputs

Figure 4
Reset Concept

3 Interfaces and Signal Description

The target of the TB macro are devices that use FPI bus for internal interfaces.

All signals are active high until otherwise specified. Active low signals are designated by "_N" (FPI mode) appended to their names. To make the design as re-usable as possible, a bus signal whose width is application dependent is specified with one of the following parameters:

Parameter name	Bus Type
CNB	Channel Number Bus
DBB	Data/Status Bus
TTC	Threshold Transmit Code
BTC	Threshold Burst Code
ITBS	Buffer Size Bus
RTL/STL	Request/Served Burst Length Bus
N	Link List Pointer Bus/ Data Buffer Address Bus
TFPIAB	TFPI address bus (MSB)

3.1 Signal Description

In the following sections, "Flexible Peripheral Interconnect (FPI) Bus compliant" means that the specified bus uses a subset of the FPI features and satisfies the basic address and data cycle. Not all FPI signals are implemented because default values are sufficient for the application i.e. they can be coded as constants in the hardware. Refer to the FPI bus specification for details of the complete bus.

The following tables lists the FPI-Bus signals and two additional out-band signals:

- TB_STAT to indicate to PMT if transferred word is status instead of pure data. Captured by PMT during data phase.
- TB_EMPTY to indicate to PMT when the TB has no data stored for requested channel.
- TB_REQ_N to indicate that at least for one channel the number of empty cells has reached the programmed burst threshold size.
- DTSTAT while data phase of DMUT to indicate if transferred word is status instead of pure data. Status bit is transferred transparently through TB to PMT.

Table 5
Macro Interfaces and Signal Description

Symbol name	I/O	Function
Clock and Reset		
SYSCLK	I	Internal system clock (66 MHz)
RESET_N	I	General reset of TB. All registers and RAM reset or initialization.
GCSTOP	I	Stop all non configuration process in TB (fast programm mode)
TB_IIP	O	Initialization of TB rams in progress
Protocol Machine Receive (PMT) Interface		
PTRD_N	I	PMT read. Only single word write transfer is supported.
PTA [CNB-1:0]	I	Address bus. Specifies channel number for transfer.
TB_PTD [DBB-1:0]	O	TB Data/Status word being transferred to PMT.
TB_PTRDY	O	Ready. End of data transfer indication. 0 => TB inserts wait state. 1 => TB will finish transfer during next clock cycle.
TB_PTSTAT	O	additional status bit
TB_PTEMPTY	O	Asserted for one cycle while PMT read if TB has no data stored for the channel specified by the address bus.
DMU Transmit (DMUT) Interface		
DTA	I	Address bus (1 bit) 0 => Request register. 1 => Data register.
DTRD_N	I	DMUT Read (of request register)
DTWR_N	I	DMUT Write (of request register or data register)
DTD[DBB-1:0]	I	Input for request register of data register

Table 5
Macro Interfaces and Signal Description (cont'd)

Symbol name	I/O	Function
DTSTAT	I	Status indication from DMUT
TB_DTREQ_N	O	Service request from TB to DMUT controller. Asserted as long as TBAQ is not empty.
TB_DTD[DBB-1:0]	O	Data out. Request register from TB to DMUT
TB_DTRDY	O	Ready. End of data or command transfer indication. 0 => TB inserts wait state. 1 => TB will finish transfer during this clock cycle.

SMIF interface

BPI_DATA[DBB-1:0]	I	BPI data input
BPI_RD_SFR_N[3:0]	I	BPI read signals
BPI_WR_SFR_N[3:0]	I	BPI write signals
BPI_REQ_N	I	BPI request (asserted with read or write)
TB_BPI_DATA[DBB-1:0]	O	BPI data output
TB_BPI_RDY_N	O	BPI ready (asserted in data cycle of read/write access)

FPI Slave Interface (alternatively to SMIF interface)

TFPI_SEL_N	I	Slave select.
TFPI_A[TFPIAB-1:2]	I	Address bus.
TFPI_D[DBB-1:0]	I	Input Data. Active during data phase of write cycle.
TFPI_WR_N	I	TFPI write to TB
TFPI_RD_N	I	TFPI read TB
TFPI_RDY	I	TFPI ready input
TB_TFPI_RDY	O	Ready. End of transfer indicator: 0 => Master should insert wait states 1 => TB will complete transfer in this cycle
TB_TFPI_RDY_EN	O	RDY output enable
TB_TFPI_D[DBB-1:0]	O	Output Data. Active during data phase of write cycle.

Table 5
Macro Interfaces and Signal Description (cont'd)

Symbol name	I/O	Function
TB_TFPI_D_EN	O	D output enable

Interrupt Controller (IC) Interface

ICTBGNT_N	I	Grant Line
TB_ICREQ_N	O	Request Line
TB_ICD[DBB-1:0]	O	TB interrupt vector data
TB_ICD_EN	O	TB interrupt vector data enable

3.2 Data Flow and Functional Timing

3.2.1 TB Interface to the Protocol Machine Transmit (PMT)

The PMT initiates an address cycle by asserting the read signal PTRD_N. TB captures the channel number from the address bus PTA during this cycle. During the data phase, PMT captures the data as soon as possible. TB asserts TB_PMTRDY during the clock cycle in which it can complete the data transfer.

Two out-of-band signals are required

1.TB_STAT to indicate if transferred word is status instead of pure data. Captured by PMT during TB_PTRDY.

2.TB_EMPTY to show PMT when the TB has no channel data stored. Also captured by PMT during TB_PTRDY (TB_PTD and TB_PTSTAT are not valid).

TB inserts 1 upto 4 waitstates after PTRD_N. Therefore TB can handle a PTRD_N each fifth cycle.

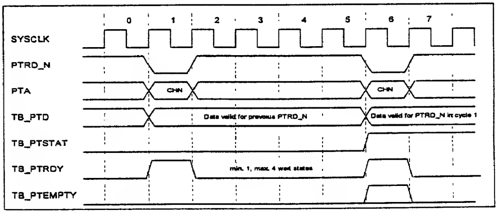

0
1
2
3
4
5
6
7
SYSCLK
PTRD_N
PTA
CHN
CHN
TB_PTD
Data valid for prevous PTRD_N
Data valid for PTRD_N in cycle 1
TB_PTSTAT
TB_PTRDY
min. 1, max. 4 wait states
TB_PTEMPTY

Figure 5
Read Access from PMT to TB

3.2.2 TB Interface to DMU Controller Receive (DMUT)

The TB interface to the DMUT is based on a bidirectional FPI slave bus. TB initiates a request register access from DMUT by activating TB_REQ_N. The read select signal SEL_N is implicitly active and not physically present. Also OPC is physically not present. DMUT sees TB as a request register at address 0 and a FIFO data port at address 1. Only 1 address bit is defined on the bus.

Two out-of-band signals are required:

1.TB_REQ to indicate that Action Queue FIFO is not empty.

2.DTSTAT to indicate if word in the burst is status instead of pure data. This signal is only valid for current data in DMU transfer.

Data request fromTB: After asserting TB_DTREQ_N DMUT initiates an address cycle (address 0). During the data cycle, TB returns the request register value (channel number and number of requested data) and asserts TB_DTRDY (possible wait states).

After preparing transmit data/status DMUT initiates an address cycle (address 0). During the data cycle DMUT writes channel information and transferlength back to request register. If last word contains frame end or abort indication CI bit is active. DMUT then transfers BL words with overlapped cycles. TB will insert wait states. In best case, a burst cycle can occur with 2 wait states but internally, the PMT and configuration interfaces have higher priority. In the following figures, a burst request from TB to DMUT (Figure5) and a data transfer from DMUT to TB is shown. TB may provide several burst requests for different and even the same channel (compare chapter , HOLD condition). DMUT (**Figure 6**) inserts several wait states after writing served burst length and channel number to request register at address 0. An ideal transfer with no collision with a PMT request (2 wait states) an one access with a collision is shown (3 wait states). TB inserts wait states by delaying TB_DTRDY.

The efficiency of the DMU transfers improves with burst length. The peak throughput (not sustainable) can approach 1 word/3 clock with long bursts. However it is ultimately limited by the PMT interface access of the internal buffer RAM. Each word transferred during a TB-DMUT data cycle inserts up to 2 wait states by TB_DTRDY.

Note that DMUT must always read and write the request register for a burst data transfer.

After setting up a new buffer, TB will request data from DMUT by it's own. This request could be used as a 'command acknowledge' indication to generate a command complete interrupt (e.g. done by DMUT). In case of a 'buffer delete' ('transmit off') command, TB discards all stored data and sends a DEL command to DMUT (RTL is not valid). This request could also be used as a 'command acknowledge' indication to generate a command complete interrupt (e.g. done by DMUT).

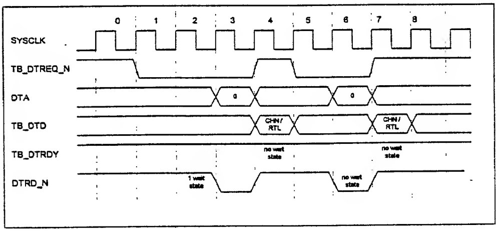


Figure 6
Burst request from TB to DMUT

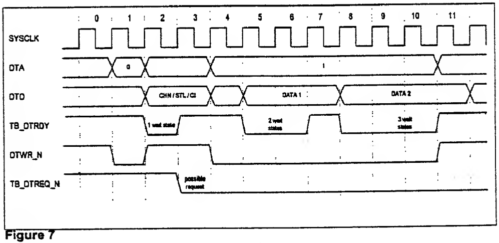


Figure 7
DMUT to TB: 2 data word transfer with wait states

3.2.3 TB Interface to FPI Configuration and Control Bus (FPI Slave) or SMIF interface

The FPI slave interface provides read and write access to all internal data and RAM. It also provides the programming interface for channelwise buffer size (ITBS) and threshold values (BTC, TTC). To change a channel configuration (buffer size and threshold) a configuration command has to be written to command register. According to the programmed values TB controller deletes or sets up a buffer. Only in case of an error condition (not enough free buffer locations for new ITBS) an interrupt is requested to DMUI. Several configuration commands can be written very fast and stored in a configuration fifo if GCSTOP is asserted. In this case, all other interfaces are deactiveated. All configuration commands are executed according to the fifo entries.

FPI Slave interface also provides system test capabilities. Each ram location can be read and written for test purposes. For reading (and writing) rams an autoincrement function is implemented: TYPE in command register (**chapter 4**) specifies the ram and the autoincrement feature interprets all data register read (write) as a read (write) data at next ram address.

The command address is implementation dependent. Figure 7 shows an implementation with command address '0' for channel number, address '1' for channel parameter ITBS, TTC and BTC.

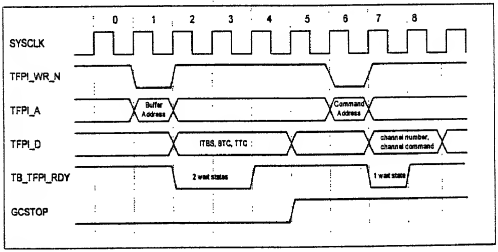


Figure 8
Configuration of a new channel

Note: TFPI interface is an 'add on'. TB macro offers a SMIF register interface with read/ write signals for all readable/writeable register. As via TFPI, a transmit init/off command is executed with a write of the channel specification command register/address, a transmit debug command is excuted with a read of channel specification buffer register after a previous write of the debug command.

3.2.4 Interrupt Controller Interface (IC)

TB activates TB_IC_REQ_N to indicate an interrupt vector on TB_IC_D. Data are valid in cycle after IC_DTGNT

Table 6
Interrupt Vector

Bit	P1	P2	.		P3			P4	P5
Function	IID		reserved		CDMF	reserved		CHN	

IID: interrupt ID (parameter)

CMDF: command fail interrupt (bit position is parameter)

CHN: channel number

P1: tb_ic_did_lb_c
P2: tb_ic_did_rb_c
P3: tb_ic_derr_c
P4: tb_ic_dchn_lb_c
P5: tb_icd_dchn_rb_c

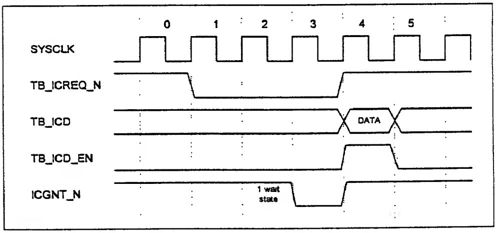


Figure 9
Interrupt Controller Interface

4 Register Description

4.1 Register Overview

Registers have to be provided to configure the channels (FIFO size, threshold value and channel number) and to read and write the internal RAMs (indirect access). and channel parameters.

4.2 Detailed Register Description

Configuration registers can be gathered togehter (TTC, BTC, ITBS, CHN, CFM in one single register) or divided in several registers implementation dependent. Bit positions are also implementation dependent.

Following solution shows 5 register solution; with each write of the channel command register, a new configuration command is written to a configuration FIFO in TB.

Note 1: A new configuration is started with write of channel command register. To programm all channels in the same manner, only for one channel the TTC, BTC, ... registers have to be written!

Note 2: After a successfull 'buffer create' command (i.e. buffer exists) first a 'buffer delete' command' has to be given, before the next 'buffer create' can be programmed. For details see chapter 2.

Note 3: 'P' indicates that this bit position/ bus width is a parameter.

4.2.1 Channel Command Register

Access	: write
Address	: *tb_vcs_cmd_adn_c*
Reset Value	: 00000000_H

	P1	P2	P3		P4 P5
Reserved	init	off	debug	Reserved	CHN

CHN (number of bits and position are parameters):

channel number

CMD (number of bits and position are parameters):

channel command: TB provides 3 commands

- buffer create ('transmit init'),
- buffer delete ('transmit off'),
- buffer parameter debug ('transmit debug') command.

P1: tb_vcs_init_c

P2: tb_vcs_off_c
P3: tb_vcs_debug_c
P4: tb_vcs_chn_lb_c
P5: tb_vcs_chn_rb_c

4.2.2 Buffer Parameter Register

Access	: write/read
Address	: *tb_cvs_bufs_adn_c*
Reset Value	: 00000000_H

P1 P2 P3 P4 P5 P P6

reserved	TTC	reserved	BTC	reserved	ITBS

BTC (number of bits and position are parameters):

Burst Threshold Code.

0: burst threshold = 1 word

others: burst threshold = $2^{(BTC+1)}$ words (as an example, coding is also a parameter)

TTC (number of bits and position are parameters):

Transmit Threshold Code.

0: burst threshold = 1 word

others: burst threshold = $2^{(TTC+1)}$ words (as an example, coding is also a parameter

ITBS (number of bits and position are parameters):

Individual Transmit channel Buffer Size = ITBS words

P1: tb_tb_vcs_ttc_lb_c
P2: tb_tb_vcs_ttc_rb_c
P3: tb_tb_vcs_btc_lb_c
P4: tb_tb_vcs_btc_rb_c
P5: tb_tb_vcs_itbs_lb_c
P6: tb_tb_vcs_itbs_rb_c

4.2.3 Indirect Access Register: Address

All rams and some register can be read/written by writing 'indirect access register address'. If autoincrement function is selected, with each read/ write access to 'indirect access register data' the next ram address is read/written. Only a startaddress has to be defined. Autoincrement may be stopped (no read/write) or interrupted (by new TYPE definition).

Access	: read/write

Address : *tb_tac_adn_c*

Reset Value : 00000000$_H$

P1 P2		P3		P4 P5
MID	reserved	AINC	reserved	COMMAND

P6 P7
Channel#/Address

MID:

Macro Idenitfication Number; access defined by TYPE etc. is only performed if MID matches the (implementation specific) ID.

AINC:

Automatic Incrementation

AINC=1 : The address will be automatically incremented with each read or write access

AINC=0 : The address won't be incremented (default value).

P1: tb_vg_mid_lb_c
P2: tb_vg_mid_rb_c
P3: tb_vg_ai_c
P4: tb_vg_cmd_lb_c
P5: tb_vg_cmd_rb_c
P6: tb_vg_adr_lb_c
P7: tb_vg_adr_rb_c

COMMAND:

tb_vg_cmd_fpc_c: read free pool counter
tb_vg_cmd_fpp_c: read free pool pointer
tb_vg_cmd_gfpc_c: read global free pool counter
tb_vg_cmd_aqctr_c: read action queue counter
tb_vg_cmd_cqctr_c: read configuration queue counter
tb_vg_cmd_pmt_on_c: switch pmt interface on
tb_vg_cmd_pmt_off_c: switch pmt interface off
tb_vg_cmd_pmt_c: switch to pmt interface mode
tb_vg_cmd_dmut_on_c: switch dmut interface on
tb_vg_cmd_dmut_off_c: switch dmut interface off
tb_vg_cmd_dmut_req_c: select dmut request register
tb_vg_cmd_dmut_dat_c: select dmut data register
tb_vg_cmd_aq_c: read/write action queue ram

tb_vg_cmd_cq_c: read/write configuration queue ram
tb_vg_cmd_pt0_c: read/write parameter table ram
tb_vg_cmd_pt1_c: read/write parameter table ram
tb_vg_cmd_pt2_c: read/write parameter table ram
tb_vg_cmd_pt3_c: read/write parameter table ram
tb_vg_cmd_ll_c: read/write link list ram
tb_vg_cmd_db_c: read/write data buffer ram
tb_vg_cmd_stat_c: read/write status bit ram

Channel#/Address:

Channel number or Address field

4.2.4 Indirect Access Register: Data

Access	: read/write
Address	: *tb_td_adn_c*
Reset Value	: 00000000_H

31	16
Test Data	

15	0
Test Data	

Test Data:

Data from/for different Rams / Free Pool Counter (number of unused buffer locations).

6 Appendix: M256F tb_shell

For M256F application a special naming convention is used. Addtional 'daisy chain' signals are added.

6.1 Register Description

6.1.1 Register Overview

Table 7PCI Slave Register Set (Direct Addressing)

Register Name		Read/ Write	Offset to PCI BAR1
Virtual Global Registers			
	TAC	R/W	058_H
	TD	R/W	$05C_H$
Macro Specific Registers			
Virtual Channel Specification Registers			
	CSPEC_CMD	W	000_H
	CSPEC_BUFFER	R/W	020_H

6.1.2 Detailed Register Description

6.1.2.1 (Virtual) Channel Specification Command (CSPEC_CMD)

Access	: write/read (only CHAN readable)
Address	: 000_H
Reset Value	: 00000000_H

31	16
CMD_XMIT(7:0)	CMD_REC(7:0)

15 8	7 0
00	CHAN(7:0)

Note: The Virtual Channel Spec (VCS) Command Register has to be programmed after all other required VCS registers, in order to initiate the programming of all macros. In case of a debug command, first the command register has to be written, then a broadcast read of the virtual channelspec is possible by read of all VCS data registers. Only the macro which has implemented the corresponding bit has to drive the register bit, otherwise drives '0' to provide broadcast read feature.

CHAN(7:0): selected channel number to be programmed

CMD_XMIT(7:0): for details refer to table "Command Description"

Note: Transmit Init for a channel must be programmed only after reset or after a Transmit Off command, i.e. two Transmit Init commands for the same channel are not allowed

Command Table Transmit:

31	30	29	28	27	26	25	24	
Rsvd	Rsvd	Idle	Debug	HOLD RESET	ABORT	OFF	INIT	function
0	0	0	0	0	0	0	1	transmit init
0	0	0	0	0	0	1	0	transmit off
0	0	0	1	0	0	0	0	transmit debug
0	0	0	0	0	0	0	0	transmit nop

6.1.2.2 (Virtual) Channel Specification Buffer (CSPEC_BUFFER)

Access	: read/write
Address	: 020_H
Reset Value	: 00000000_H

31 29	28 16
TQUEUE(2.0)	ITBS(12.0)

15 12		10 8		6 4		2 0
TTC(3:0)	0	TBTC(2:0)	0	~~RQUEUE(2:0)~~	0	~~RBTC(2:0)~~

TBTC: transmit burst threshold code

ITBS: individual transmit buffer size

TTC: Transmit Threshold Code

(for coding see chapter 6.4)

6.1.3 Test Command Register (TAC)

Access	: read/write
Offset Address	: $_H$
Reset Value	: 00000000$_H$

31 28				24	23 16
MID	0	0	0	AI	CMD

15			12 0
0	0	0	ADDRESS

MID: Macro ID Code

AI: Auto Increment Function

Address: internal address

CMD: Command (select of ram and register)

CMD:

23	22	21	20	19	18	17	16	
								function
0	0	0	0	0	0	0	0	tb_vg_cmd_fpc_c
0	0	0	0	0	0	0	1	tb_vg_cmd_fpp_c
0	0	0	0	0	0	1	0	tb_vg_cmd_gfpc_c
0	0	0	0	0	0	1	1	tb_vg_cmd_aqctr_c
0	0	0	0	0	1	0	0	tb_vg_cmd_cqctr_c
1	0	1	0	1	1	1	0	tb_vg_cmd_pmt_on_c
1	0	1	0	1	0	1	0	tb_vg_cmd_pmt_off_c
1	0	1	0	0	0	0	0	tb_vg_cmd_pmt_c

23	22	21	20	19	18	17	16	
1	0	1	1	1	1	1	0	tb_vg_cmd_dmut_on_c
1	0	1	1	1	0	1	0	tb_vg_cmd_dmut_off_c
1	0	1	1	0	0	0	1	tb_vg_cmd_dmut_req_c
1	0	1	1	0	0	0	0	tb_vg_cmd_dmut_dat_c
1	1	1	1	0	0	0	0	tb_vg_cmd_aq_c
1	1	1	1	0	0	0	1	tb_vg_cmd_cq_c
1	1	1	1	0	0	1	0	tb_vg_cmd_pt0_c
1	1	1	1	0	0	1	1	tb_vg_cmd_pt1_off_c
1	1	1	1	0	1	0	0	tb_vg_cmd_pt2_c
1	1	1	1	1	0	0	0	tb_vg_cmd_pt3_c
1	1	1	1	0	1	0	1	tb_vg_cmd_ll_c
1	1	1	1	0	1	1	0	tb_vg_cmd_db_c
1	1	1	1	0	1	1	1	tb_vg_cmd_stat_c

Macro ID Code:

31	30	29	28	macro
0	0	1	1	TB (tb_vg_mid_tb_c)

General

- the test access provides read/write access of important internal rams and registers
- test registers are virtuals global registers (SEL signal: pb_vg_tfpi_sel_n / implemented as a daisy chain).
- the single macros are selected by the MID code of the test command register
- CMD specifies/selects one of the macro rams/registers
- the address field is used to access a ram address
- AI: autoincrement : address given in the address field is incremented automatically for each access
- All macros which are not selected by MID drive data output "00000000" and <macro>_TPFI_RDY = '1'. Driving "00000000" would mean not disable the enable line for data out, but to set the output data to "00000000".

Test write access:

1. Write TAC
2. Write TAD

Test read access:

1. Write TAC
2. Read TAD

Typically the macro selected via MID delays the RDY signal until the selected ram/register has been read and the data can be provided at the TFPI interface. No prefetch of testdata is required.

Note: CMD/Address have to be defined for each macro; there is no read/write selection in the CMD field; rd/wr's are handled with the TFPI read & write signals

6.1.4 Test Data Register (TD)

Access	: read/write
Address	: $5C_H$
Reset Value	: 00000000_H

31	
TEST DATA	

15	0
TEST DATA	

TEST DATA:	ram/register test data (read or write)

6.2 DMUT interface

Table 8
Request Register for data request from TB to DMUT

Bit	31	·		28	16			7	0
Function	DEL	Reserved		RTL (requested transfer length)		Reserved		CHN (Channel number)	

Table 9
Request Register for data transfer acknowledge from DMUT to TB

Bit	31			28	16			7	0 -
Function	CI	Reserved		STL (served transfer length)		Reserved		CHN (Channel number)	

6.3 Interrupt Controller Interface (IC)

Table 10
Interrupt Vector

Bit	31	28	.		17			7	0
Function	1001		0		CDMF	0		CHN	

6.4 Ram sizes

6.4.1 TB_PTR (TB_PTR1)

Parameter Table: 256 x 89

Table 11: All channel parameters:

ITBS	BTC	TTC	FIRST	AQE	WAIT	CIPV	CIP	FCTR	ECTR	WP	RP
(13)	(3)	(4)	(1)	(1)	(1)	(1)	(13)	(13)	(13)	(13)	(13)

6.4.2 TB_DBR and TB_LLR

Data Buffer: 8K x 33

Link List: 8K x 13

6.4.3 TB_AQR

Action Queue: 256 x 8

6.4.4 TB_CFGR (TB_RTR2)

Configuration Queue: 256 x 28

Table 12: All locations:

ITBS	BTC	TTC	CHN
(13)	(3)	(4)	(8)

6.5 Shell Interface and Signal Description

M256F Symbol name	I/O	Macro Symbol Name	Function
Clock and Reset			
SYSCLK	I	SYSCLK	Internal system clock (33 MHz)
HW_RESET_N	I	(RESET_N)	General hardware reset of TB. All registers and RAM reset or initialization.
SW_RESET_N	I	-	General sofrware reset of TB. All registers and RAM reset or initialization.
SCANMODE	I	-	Scanmode
GC_STOP	I	GCSTOP	Stop all non configuration process in TB (fast programm mode)
TB_IIP	O	TB_IIP	Initialization of TB rams in progress
Protocol Machine Receive (PMT) Interface			
PT_TB_RD_N	I	PTRD	PMT read. Only single word write transfer is supported.
PT_TB_A [CNB-1:0]	I	PTA	Address bus. Specifies channel number for transfer.
TB_PT_D [DBB-1:0]	O	TB_PTD	TB Data/Status word being transferred to PMT.
TB_PT_RDY	O	TB_PTRDY	Ready. End of data transfer indication. 0 => TB inserts wait state. 1 => TB will finish transfer during next clock cycle.
TB_PT_STAT	O	TB_PTSTAT	Mode of data word to be transferred. 0 => protocol data 1 => status + protocol data

M256F Symbol name	I/O	Macro Symbol Name	Function
TB_PT_EMPTY	O	TB_PTEMPTY	Asserted for one cycle while PMT read if TB has no data stored for the channel specified by the address bus.

DMU Transmit (DMUT) Interface

DT_TB_RD_N	I	DTRD	Read. DMUT Read Control (for request register)
DT_TB_WR_N	I	DTWR	Write. DMUT Write Control (for request register or data register)
DT_TB_A	I	DTA	Address bus (1 bit) 0 => Request register. 1 => Data register.
DT_TB_D[DBB-1:0]	I	DTD	Data in. Input command from DMUT (channel number e. g.) or data/ status input
DT_TB_STAT	I	DTSTAT	Status indication from DMUT
TB_DT_REQ_N	O	TB_DTREQ	Service request from TB to DMUT controller. Asserted as long as TBAQ is not empty.
TB_DT_D[DBB-1:0]	O	TB_DTD	Data out. Request register from TB to DMUT controller
TB_DT_RDY	O	TB_DTRDY	Ready. End of data or command transfer indication. 0 => TB inserts wait state. 1 => TB will finish transfer during this clock cycle.

Interrupt Controller (IC) Interface

M256F Symbol name	I/O	Macro Symbol Name	Function
TB_IC_REQ_N	O	TB_ICREQ	Request Line
IC_TB_GNT_N	I	ICTBGNT	Grant Line
TB_IC_D[31:0]	O	TB_ICD	TB interrupt vector data
IC_D	I	-	daisy chain data input

FPI Slave Interface

PB_TFPI_RD_N	I	TFPI_RD_N	TFPI read TB
PB_TFPI_WR_N	I	TFPI_WR_N	TFPI write to TB
PB_TFPI_A[8:2]	I	TFPI_A[TFPIAB-1:2]	Address bus.
PB_TFPI_D[31:0]	I	TFPI_D	Input Data. Active during data phase of read cycle.
PB_TFPI_RDY	I	TFPI_RDY	TFPI ready input
PB_VC_TFPI_SEL_N	I	TFPI_VC_SEL_N	Slave select.
PB_VG_TFPI_SEL_N	I	TFPI_VG_SEL_N	Slave select.
TFPI_D[31:0]	O	-	Daisy chain input.
TB_TFPI_RDY	O	TB_TFPI_RDY	End of transfer indicator: 0 => Master should insert wait states 1 => TB will complete transfer in this cycle
TB_TFPI_D[31:0]	O	TB_TFPI_D	Output Data. Active during data phase of write cycle.

6.6 Dimension of M256F application:

Name	DescriptionO	M256F value
tb_chn_nw_c	number of supported channels	256
tb_chn_nb_c	channel bus bits	8
tb_db_nw_c	depth of data buffer	8192
tb_dba_nb_c	data buffer address bus bits	13
tb_btc_nb_c	number of BTC bits	3

tb_ttc_nb_c	number of TTC bits	4
tb_btc2btl	coding of TTC values	0: 1 1: 4 2: 8 3: 16 4: 32 5: 64 6: 128 7: 256
tb_ttc2ttl	coding of TTC values	0: 1 1: 4 2: 8 3: 12 4: 16 5: 24 6: 32 7: 40 8: 48 9: 64 A: 96 B: 128 C: 192 D: 256 E: 384 F: 512
tb_data_nb_c	data bus width	32
tb_test_pt*_**_c (*: 1,2,3 and **: rb,lb)	selected slices for testmode access of parameter table ram	1: lb=63/rb=32 2: lb=88/rb=64 3: lb=88/rb=64 (3 not used)

Appendix H

Title:
Macro Specification
TB data sheet

1 Introduction

1.1 Overview

The Transmit Buffer (TB) provides channelwise buffering of raw data/status words between a Data Management Unit Transmit (DMUT) and a Protocol Machine Transmit (PMT). This data is stored in form of (internal) linked lists for all logical channels. These linked lists are pre-allocated according to bandwidth requirements of the respective channels: a channelwise buffersize is allocated via a channelwise programmable parameter ITBS (Individual Transmit Buffer Size). ITBS has a granularity of 1 dword. In order to avoid transmit underrun conditions each channel buffer has two control parameters for smoothing the filling/emptying process (Transmit Threshold, Request Threshold)

The transmit enable threshold value TTC has following impact:

- Transmission of channel data is started if more than the transmit threshold number of words are he stored in the internal transmit buffer or if at least one word with a complete indication (CI bit, see interface description TB-DMUT) is stored in channel buffer. Otherwise an empty indication (TB_PTEMPTY) will be delivered to PMT. This feature is always active. Typically after a configuration command or after 'TB empty' condition, but also while frame oriented data transfer this feature reduces the probability of data underrun. Applications without of CI-feature leads to a single activation of transmit threshold feature after a 'transmit init' command. After first transmit enable transmission of data will never be interrupted through a fill level below threshold.

The burst threshold value TBTC has following impact:

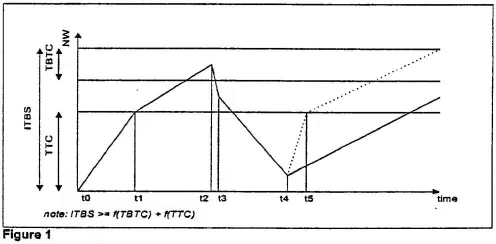
- As soon as the amount of empty channel buffer locations gets above the threshold value the TB will request data from the transmit DMU (DMUT) .

TB works mode and frame independent. TB transfers data (even the status bit) absolutely transparent and fulfills a channelwise buffering and an efficient data burst request.generating.

1.2 Features incl. performance, number of gates

- Number of gates: depends strongly on TB dimensions (see dimensions in M256F implementation in chapter 5)
- Performance: The overall limitation of data throughput is the PMT interface: each 5th cyle a read access to TB is allowed.
- SMIF register interface or alternatively Target FPI interface for configuration of TB (transmit init, off and idle commands), system test (read of programmable parameter 'burst threshold', 'transmit threshold' and 'channel buffer size' via channel debug command) and testmode (read/write of all internal rams via indirect test register access)

- Interface to DMUT and PMT is FPI like
- interrupt controller (IC) interface with channelwise interrupt generation (configuration fail interrupt, not maskable)
- Programmable FIFO size for each channel (ITBS)
- Programmable burst threshold for DMUT request generation (TBTC) and transmit enable threshold (TTC)
- implementation specific number of supported channels, complete buffer size and burst and transmit threshold codes (number of codes and codes itself).Burst and transmit thresholds are independent.



note: ITBS >= f(TBTC) + f(TTC)

Figure 1
Threshold Feature

- t0: 'transmit init' command, TB sent request to DMUT (ITBS data words); at t0 DMUT starts data transfer to TB
- t0 - t1: DMUT writes (some) data to TB; NW (number of words) is still below TTC; transmit not yet enabled
- t1: NW=TTC, TB is enabled to transfer data to PMT
- t1 - t2: DMUT is still writing data to TB, PMT reads data from TB; fill rate is slowed down
- t2: all initial (=ITBS) requests served, DMUT stops writing data to TB
- t2 - t3: PMT reads data
- t3: after NW<(ITBS-TBTC) a data request has been sent to DMUT; at t=t3 DMUT starts data transfer back to DMUT
- t3 - t4: PMT reads more data than DMUT can deliver; NW decreases;

- t4 - t5 (dotted line): at t4 a new frame would start being transmitted to PMT, but NW<TTC; therefore transmit is again disabled until NW=TTC; at t5 NW=TTC and data are transferred to PMT. DMUT fills TB (DMUT write is faster than PMT read)
- after t4 (line): in case of no complete indication (no frame end) data will be still transfered to PMT, the data fill up rate is smaller than in "dotted" case.

1.3 System IntegrationSystem Integration and Application

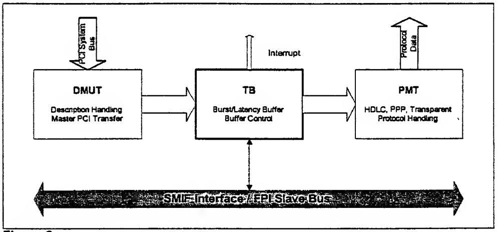


Figure 2
System Integration

The TB has four interfaces:

- SMIF / FPI Slave Interface for programming and system test
- DMUT interface for burst capable data transfer from external memory
- PMT interface for protocoll interface
- DMUI interrupt interface

1.4 Known Restrictions and Problems

The maximum data throughput depends on data bus width, system clock frequency and the programmed configuration of TB.

PMT limitation: The maximum speed of PMT interface is 1 data transfer per 5 cycles. This is a short term maximum througput. TB is able to handle an average througput of 1 data per 20 clock cycles.

With higher burst thresholds and less number of activated channels this average increases.

A further limitation is the behaviour of TB-DMUT-interface: DMUT allows no interleaving of burst request and data transfer, i.e. first a read of request register is done, then either the data are served or the request is stored in DMUT (e.g. HOLD condition in DMUT, see DMUT specification). Afterwards the read of request register is done.

2 Functional and Test Description

2.1 Block Diagram

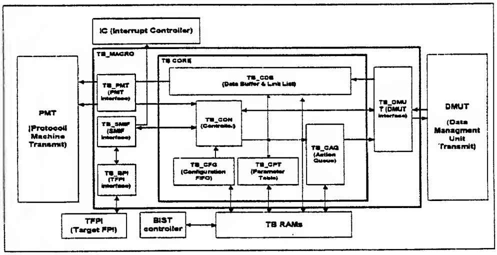


Figure 3
TB Block Diagram

As described in the **Figure 3**, the Transmit Buffer consists of

- Central Data Buffer and link list (TB_CDB)
- Central Parameter Table (TB_CPT) and configuration FIFO (TB_CFG)
- Central Action Queue (TB_CAQ)
- Controller (TB_CON).

Those blocks build the core functionality. Aditionally TB has a FPI like DMUT (TB_DMUT) and a FPI like PMT (TB_PMT) interface. TB macro has an internal SMIF interface (TB_SMIF). Therefore TB offers both

- SMIF and
- TFPI interface (TB_BPI implementation is an optional part of TB Macro).

2.2 Normal Operation Description

After reset all rams are initialized: each ram address is written with idle values and a global link list is set. If initialisation is ready, TB asserts TB_IIP inactive (Duration of initialisation depends on size of largest ram, typically data buffer is ram with highest depth: number of words is equal duration in cycles).

Afterwards TB could be configured via TFPI/SMIF interface: a channel command consists of 2 write accesses (TB point of view): first the channel specification buffer register (CSPEC_BUFFER) has to be written with the buffer parameters, then a write of the channel specification command register (CSPEC_CMD) specifies the channel number and the desired command (init, off).

All commands are written into a configuration fifo. This fifo can't be read before GC_STOP is deasserted. Therfore TB won't generate a data request and can't transfer data to PMT during this 'stop' phase.

When GC_STOP becomes inactiv, the configuration fifo is read.

A transmit init command leads to an allocation of the requested buffer size (ITBS) and writes the according channel number into action queue. When this action queue is read a data request to DMUT is asserted. After an init command, TB always requests a number of ITBS data from DMUT. Later, the number of requested data will be a function of TBTC, i.e. TB will always request equal or more then TBTC data. When 'transmit threshold' condition is fullfilled, TB delivers data according to a PMT read access. Otherwise TB sets an empty indication (TB_EMPTY).

A transmit off command deactivates the channel and discards all stored channel data. The previous allocated buffer locations become available. The next request to DMUT indicates with DEL=1, that this channel is switched off and all stored requests in DMUT have to be cancelled. RTL has to be ignored and no further requests are generated. All further PMT reads will be acknowledged by TB_EMPTY.

As long as no or less then TTC data are stored in TB, each PMT request is answered with an TB_EMPTY indication.

Incoming data from DMUT will be stored in TB data buffer (TB_DB) by using the first element from a free pool list. A read request from the PMUT causes the TB to read next data in the channel data buffer (link list) and to add the read and therefore emptied location to the free pool list.

Table 1

Request Register for data request from TB to DMUT

Bit	P1			P2	P3			P4	P5
Function	DEL	Reserved		RTL (requested transfer length)		Reserved		CHN (Channel number)	

Table 2
Request Register for data transfer acknowledge from DMUT to TB

Bit	P6			P2	P3			P4	P5
Function	CI	Reserved		STL (served transfer length)		Reserved		CHN (Channel number)	

Bitpositions, defined in tb_package:
P1: tb_dt_crdel_pb_c
P2: tb_dt_crbl_lb_c
P3:tb_dt_crbl_rb_c
P4:tb_dt_crchn_lb_c
P5:tb_dt_crchn_rb_c
P6:tb_dt_crci_pb_c

The status of each logical channel is stored in the Parameter Table TB_PT. As soon as the amount of data stored in the TB gets below the programmable threshold (buffer size minus burst size) for a specific channel, an entry to the action queue (TB_AQ) is made. TB_AQ is a FIFO which is dimensioned according to the number of channels (depth = number fo channels). For each channel only one entry is possible. Only the channel number is stored. When the next entry of TB_AQ is read, a request TB_DTREQ from TB to DMUT is genearated with the number of actual empty buffer locations of corresponding channel. Both channel number (CHN) and the number of words (RTL) are written to the request register (read access, refer to **Table 1**). After TB_DTREQ the DMUT has to read the request register and then read the corresponding number of data via external bus interface (PCI) and write them back to TB: before data transfer DMUT writes channel number (CHN) and the number of served length (STL) into request register (write access, refer to **Table 2**). If the last word of the data transfer will be a word that allows data transfer to PMT independent of number of data stored in TB, CI is activated. In some application a frame end (FE) or abort (TAB) indication could force DMUT to set CI. CI=1 enables transmission of all data stored in TB.

Unserved or uncompletely served TB requests are stored in DMUT. In HOLD condition TB will continue to request data from DMUT if the request condition is valid. The unserved requests are stored in DMUT. After transfer of the remaining data to PMT TB will become empty. Afterwards TB sets the TB_EMPTY line to PMT. PMT will decide if it is an abort condition or not (in the first case generate an underrun interrupt and in the second case send the idle character). This channel remains inactive as long as the host CPU does not request an activation of DMUT. After activation of DMUT (e.g. 'hold reset' command) all stored requests have to be served to TB. In case of no activation a 'transmit off' switches this channel to 'idle' condition and gives all reserved buffer locations free.

Status bits and flags have no impact to TB behaviour. E.g. FE (frame end), TAB (transmit abort) and BE (byte enabales) bits are transferred transparently from DMUT through TB to PMT as all other data.

Transfers to the PMT are performed as single word for each channel. The TB provides a control flag (TB_PTSTAT) indicating the word type.

2.2.1 TB Controller (TB_CON)

The central controller is devided into 3 parts:

- serial controller
- data transfer controller
- configuration controller

The *serial controller* reacts on PT_RD requests: all channel parameters are read, data buffer location is read (inclusive link list, i.e. next pointer). Data are delivered to PMT interface and next pointer is written back to parameter table. The number of empty locations is increased and the free cell is added to free pool. TB_EMPTY is asserted if no/not enough data are stored. If burst condition is fullfilled, an action queue entry is written to AQ ram. Only one entry per channel is possible in AQ ram.

The *data transfer controller* is started by a not empty AQ ram. The channel number of AQ entry is used to read all paramters for this channel: the number of actually free channel locations is transferred to DMUT via request register entry. The channel parameters are updated, i.e. number of free cells is set to 0 and written back to parameter table ram.

A DMUT write has to be initiated by a write of request register: served transfer length has to be specified, channel number and CI bit (CI=1 shows, that data transfer ends with a status word). These informations are used when updating channel parameters after writing data to data buffer. With each write of data buffer at first cell of free pool, the free pool pointer has to be updated with next pointer. In case of a burst write, the next pointer ha to be read and therefore at least 1 wait state has to be inserted.

The *configuration controller* creates/discards channel buffer (transmit init, off, debug) and controlls IIP phase. Also testmodi are controlled via this FSM.

2.2.2 TB Parameter Table (TB_PT)

The TB_PT RAM provides a control word per channel for buffer management:

- Channel burst and transmit enable threshold, buffer size
- Count of empty words in channel's buffer chain not already sent to DMUT
- Count of stored words in channel's buffer chain
- Pointers to start and end of buffer chain (DB read and write addresses)
- Complete channel buffer status and complete indication pointer

RAM size:

NumChan (NC) words X (6N + TBTB + TTC + 4) bit RAM

Table 3: All channel parameters:

ITBS (N)	TBTC (TBTB)	TTC (TTB)	FIRST (1)	AQE (1)	WAIT (1)	CIPV (1)	CIP (N)	FCTR (N)	ECTR (N)	WP (N)	RP (N)

ITBS: individual transmit buffer size

total buffer size reserved for channel. Max size is ($2^{(N)}$-1) entries, min. size is 1 entry. Summary of all buffers is DBS entries.

TBTC/TTC: transmit burst threshold code/transmit enable code

FIRST: wait for first data after configuration

The first data entry after configuration has special conditions

AQE: action queue entry

action queue entry (only one entry per channel allowed);

WAIT: transmission wait until transmit enabled

Channel start of transmit disabled until threshold reached or one complete frame is stored in channel buffer.

CIPV: pointer to last data with a complete indication flag (CI) is valid.

CIP: pointer to last data with a complete indication flag (CI).

FCTR: full counter

number of filled locations in channel buffer. If FCTR = 0 an TB_EMPTY indication is set.

ECTR: empty counter

number of empty locations in channel buffer. If ECTR is bigger than threshold, an action queue entry has to be made

RP: read pointer

Pointer for PMT data requests; RP shows next location to be read.

WP: write pointer

Pointer for DMUT data transfers; WP shows to last written location.

TB_PT block also includes a free pool pointer (**FPP**) and a free pool counter (**FPC**).

note: N is dependent on number of words DBS in the DB RAM.

$2^{N-1} < DBS <= 2^{N}$

2.2.3 TB Data Buffer and link list (TB_DB)

The TB_DB size (DBS) is influenced by:

- number of channels supported
- sum of channel bit rates
- thresholds
- maximum bus latency

The data buffer locations are connected to a logical buffer by a pointer link list. The link list has the width N and the depth DBS. Each entry specifies the next data buffer element.

2.2.4 TB Action Queue (TB_AQ)

FIFO to buffer requests from TB to the DMU Transmit. An entry specifies only the channel number:

Size: number of channels x channel bus width

2.2.5 TB Configuration (TB_CFG)

FIFO to buffer the configuration requests from TFPI/SMIF,

Table 4: All locations:

ITBS (N)	TBTC (TBTB)	TTC (TTB)	CHN (CHN)

A buffer create command forces an entry with ITBS>0, a buffer delete command forces an entry with ITBS=0.

Size: number of channels x (N + TBTB + TTB + CHN)

ITBS: individual transmit buffer size

total buffer size reserved for channel. Max size is ($2^{(N)}$-1) entries, min. size is 1 entry. Summary of all buffers is DBS entries.

TBTC/TTC: burst threshold code/transmit enable code

CHN: channel number

2.3 Reset Behavior

The TB is reset with the line RESET_N.

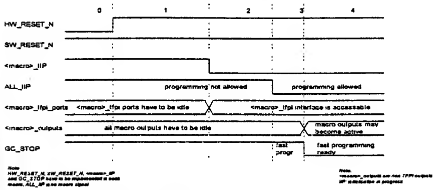
All internal registers are reset while reset condition; after reset all rams are initialized to known states (RESET state), while TB_IIP is active. The initialisation time depends on the complete data buffer size, because an initial link list in the link list ram has be to generated. During this time it is not possible to write via TFPi to TB registers. It is only allowed to write to non TB implemented TFPi registers.

After initialisation TB_IIP becomes inactiv and transmit buffer TB is ready for configuration and data transfer.

After reset/initialisation all interfaces (PMT, DMUT, IC) are inactive and the registers are accessible via the TFPI bus / SMIF for configuration.

For a fast configuration the line GCSTOP can be activated to suppress all other interface communication. If GCSTOP is asserted only TFPI interface is active and TB creates no TB_DTREQ and delivers no data in case of a PMT_RD_N (TB delivers TB_EMPTY).

Initialisation of Toplevel (macro view)



Note
HW_RESET_N, SW_RESET_N, <macro>_IIP and GC_STOP have to be implemented in each macro. ALL_IIP is no macro signal

Note.
<macro>_outputs are non TFPI outputs
IIP: initialisation in progress

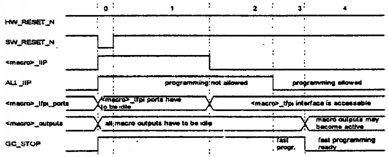
0: HW_RESET_N is active and deactivates SW_RESET_N; all macros are reset; GC_STOP set to '1'

1. HW_RESET_N inactive => each macro executes the macro specific initialisation of rams

2: some macros are ready with initialisation of rams, some not (ALL_IIP is a or-function of all <macro>_IIPs). The "ready macros" could be accessed by TFPI interface, but software programming is not allowed as long as ALL_IIP is active

3. all macros are ready with initialisation => ALL_IIP becomes inactive, software polls the ALL_IIP bit after HW_RESET_N; if ALL_IIP = 0 then software programms all macros (in a default "fast programming mode", GC_STOP = 1; reset value of GC_STOP is '1', but can be reset at any time)

4: after fast programming macro outputs (non TFPI) may become active and macro has to react on non FPI inputs



0: SW_RESET_N becomes active; all macros (registers) are reset; GC_STOP is active

1: SW_RESET_N inactive => each macro executes the mauro specific initialisation of rams

2: some macros are ready with initialisation of rams, some not (ALL_IIP is a or-function of all <macro>_IIPs). The "ready macros" could be accessed by TFPI interface, but software programming is not allowed as long as ALL_IIP is active

3. all mauros are ready with initialisation => ALL_IIP becomes inactive: software polls the ALL_IIP bit after SW_RESET_N; if ALL_IIP = 0 then software programms all macros in a "fast programming mode", GC_STOP = 1

4: after fast programming mucro outputs (non TFPI) may become active and each macro has to react on non FPI inputs

Figure 4
Reset Concept

3 Interfaces and Signal Description

The target of the TB macro are devices that use FPI bus for internal interfaces.

All signals are active high until otherwise specified. Active low signals are designated by "_N" (FPI mode) appended to their names. To make the design as re-usable as possible, a bus signal whose width is application dependent is specified with one of the following parameters:

Parameter name	Bus Type
CNB	Channel Number Bus
DBB	Data/Status Bus
TTC	Threshold Transmit Code
TBTC	Threshold Burst Code
ITBS	Buffer Size Bus
RTL/STL	Request/Served Burst Length Bus
N	Link List Pointer Bus/ Data Buffer Address Bus
TFPIAB	TFPI address bus (MSB)

3.1 Signal Description

In the following sections, "Flexible Peripheral Interconnect (FPI) Bus compliant" means that the specified bus uses a subset of the FPI features and satisfies the basic address and data cycle. Not all FPI signals are implemented because default values are sufficient for the application i.e. they can be coded as constants in the hardware. Refer to the FPI bus specification for details of the complete bus.

The following tables lists the FPI-Bus signals and two additional out-band signals:

- TB_STAT to indicate to PMT if transferred word is status instead of pure data. Captured by PMT during data phase.
- TB_EMPTY to indicate to PMT when the TB has no data stored for requested channel.
- TB_REQ_N to indicate that at least for one channel the number of empty cells has reached the programmed burst threshold size.
- DTSTAT while data phase of DMUT to indicate if transferred word is status instead of pure data. Status bit is transferred transparently through TB to PMT.

Table 5
Macro Interfaces and Signal Description

Symbol name	I/O	Function

Clock and Reset

SYSCLK	I	Internal system clock (66 MHz)
RESET_N	I	General reset of TB. All registers and RAM reset or initialization.
GCSTOP	I	Stop all non configuration process in TB (fast programm mode)
TB_IIP	O	Initialization of TB rams in progress

Protocol Machine Receive (PMT) Interface

PTRD_N	I	PMT read. Only single word write transfer is supported.
PTA [CNB-1:0]	I	Address bus. Specifies channel number for transfer.
TB_PTD [DBB-1:0]	O	TB Data/Status word being transferred to PMT.
TB_PTRDY	O	Ready. End of data transfer indication. 0 => TB inserts wait state. 1 => TB will finish transfer during next clock cycle.
TB_PTSTAT	O	additional status bit
TB_PTEMPTY	O	Asserted for one cycle while PMT read if TB has no data stored for the channel specified by the address bus.

DMU Transmit (DMUT) Interface

DTA	I	Address bus (1 bit) 0 => Request register. 1 => Data register.
DTRD_N	I	DMUT Read (of request register)
DTWR_N	I	DMUT Write (of request register or data register)
DTD[DBB-1:0]	I	Input for request register of data register

Table 5
Macro Interfaces and Signal Description (cont'd)

Symbol name	I/O	Function
DTSTAT	I	Status indication from DMUT
TB_DTREQ_N	O	Service request from TB to DMUT controller. Asserted as lóng as TBAQ is not empty.
TB_DTD[DBB-1:0]	O	Data out. Request register from TB to DMUT
TB_DTRDY	O	Ready. End of data or command transfer indication. 0 => TB inserts wait state. 1 => TB will finish transfer during this clock cycle.

SMIF interface

BPI_DATA[DBB-1:0]	I	BPI data input
BPI_RD_SFR_N[3:0]	I	BPI read signals
BPI_WR_SFR_N[3:0]	I	BPI write signals
BPI_REQ_N	I	BPI request (asserted with read or write)
TB_BPI_DATA[DBB-1:0]	O	BPI data output
TB_BPI_RDY_N	O	BPI ready (asserted in data cycle of read/write access)

FPI Slave Interface (alternatively to SMIF interface)

TFPI_SEL_N	I	Slave select.
TFPI_A[TFPIAB-1:2]	I	Address bus.
TFPI_D[DBB-1:0]	I	Input Data. Active during data phase of write cycle.
TFPI_WR_N	I	TFPI write to TB
TFPI_RD_N	I	TFPI read TB
TFPI_RDY	I	TFPI ready input
TB_TFPI_RDY	O	Ready. End of transfer indicator: 0 => Master should insert wait states 1 => TB will complete transfer in this cycle
TB_TFPI_RDY_EN	O	RDY output enable
TB_TFPI_D[DBB-1:0]	O	Output Data. Active during data phase of write cycle.

Table 5
Macro Interfaces and Signal Description (cont'd)

Symbol name	I/O	Function
TB_TFPI_D_EN	O	D output enable

Interrupt Controller (IC) Interface

ICTBGNT_N	I	Grant Line
TB_ICREQ_N	O	Request Line
TB_ICD[DBB-1:0]	O	TB interrupt vector data
TB_ICD_EN	O	TB interrupt vector data enable

3.2 Data Flow and Functional Timing

3.2.1 TB Interface to the Protocol Machine Transmit (PMT)

The PMT initiates an address cycle by asserting the read signal PTRD_N. TB captures the channel number from the address bus PTA during this cycle. During the data phase, PMT captures the data as soon as possible. TB asserts TB_PMTRDY during the clock cycle in which it can complete the data transfer.

Two out-of-band signals are required

1.TB_STAT to indicate if transferred word is status instead of pure data. Captured by PMT during TB_PTRDY.

2.TB_EMPTY to show PMT when the TB has no channel data stored. Also captured by PMT during TB_PTRDY (TB_PTD and TB_PTSTAT are not valid).

TB inserts 1 upto 4 waitstates after PTRD_N. Therefore TB can handle a PTRD_N each fifth cycle.

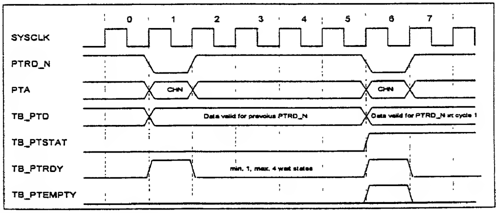

0
1
2
3
4
5
6
7
SYSCLK
PTRD_N
PTA
TB_PTD
TB_PTSTAT
TB_PTRDY
TB_PTEMPTY
CHN
CHN
Data valid for prevoius PTRD_N
Data valid for PTRD_N at cycle 1
min. 1, max. 4 wait states

Figure 5
Read Access from PMT to TB

3.2.2 TB Interface to DMU Controller Receive (DMUT)

The TB interface to the DMUT is based on a bidirectional FPI slave bus. TB initiates a request register access from DMUT by activating TB_REQ_N. The read select signal SEL_N is implicitly active and not physically present. Also OPC is physically not present. DMUT sees TB as a request register at address 0 and a FIFO data port at address 1. Only 1 address bit is defined on the bus.

Two out-of-band signals are required:

1.TB_REQ to indicate that Action Queue FIFO is not empty.

2.DTSTAT to indicate if word in the burst is status instead of pure data. This signal is only valid for current data in DMU transfer.

Data request fromTB: After asserting TB_DTREQ_N DMUT initiates an address cycle (address 0). During the data cycle, TB returns the request register value (channel number and number of requested data) and asserts TB_DTRDY (possible wait states).

After preparing transmit data/status DMUT initiates an address cycle (address 0). During the data cycle DMUT writes channel information and transferlength back to request register. If last word contains frame end or abort indication CI bit is active. DMUT then transfers BL words with overlapped cycles. TB will insert wait states. In best case, a burst cycle can occur with 2 wait states but internally, the PMT and configuration interfaces have higher priority. In the following figures, a burst request from TB to DMUT (**Figure5**) and a data transfer from DMUT to TB is shown. TB may provide several burst requests for different and even the same channel (compare chapter , HOLD condition). DMUT (**Figure 6**) inserts several wait states after writing served burst length and channel number to request register at address 0. An ideal transfer with no collision with a PMT request (2 wait states) an one access with a collision is shown (3 wait states). TB inserts wait states by delaying TB_DTRDY.

The efficiency of the DMU transfers improves with burst length. The peak throughput (not sustainable) can approach 1 word/3 clock with long bursts. However it is ultimately limited by the PMT interface access of the internal buffer RAM. Each word transferred during a TB-DMUT data cycle inserts up to 2 wait states by TB_DTRDY.

Note that DMUT must always read and write the request register for a burst data transfer.

After setting up a new buffer, TB will request data from DMUT by it's own. This request could be used as a 'command acknowledge' indication to generate a command complete interrupt (e.g. done by DMUT). In case of a 'buffer delete' ('transmit off') command, TB discards all stored data and sends a DEL command to DMUT (RTL is not valid). This request could also be used as a 'command acknowledge' indication to generate a command complete interrupt (e.g. done by DMUT).

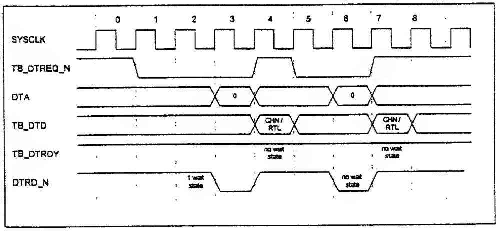


Figure 6
Burst request from TB to DMUT

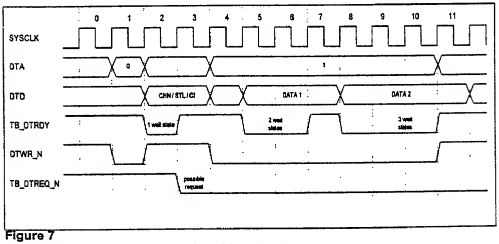


Figure 7
DMUT to TB: 2 data word transfer with wait states

3.2.3 TB Interface to FPI Configuration and Control Bus (FPI Slave) or SMIF interface

The FPI slave interface provides read and write access to all internal data and RAM. It also provides the programming interface for channelwise buffer size (ITBS) and threshold values (TBTC, TTC). To change a channel configuration (buffer size and threshold) a configuration command has to be written to command register. According to the programmed values TB controller deletes or sets up a buffer. Only in case of an error condition (not enough free buffer locations for new ITBS) an interrupt is requested to DMUI. Several configuration commands can be written very fast and stored in a configuration fifo if GCSTOP is asserted. In this case, all other interfaces are deactiveated. All configuration commands are executed according to the fifo entries.

FPI Slave interface also provides system test capabilities. Each ram location can be read and written for test purposes. For reading (and writing) rams an autoincrement function is implemented: TYPE in command register (**chapter 4**) specifies the ram and the autoincrement feature interprets all data register read (write) as a read (write) data at next ram address.

The command address is implementation dependent. Figure 7 shows an implementation with command address '0' for channel number, address '1' for channel parameter ITBS, TTC and TBTC.

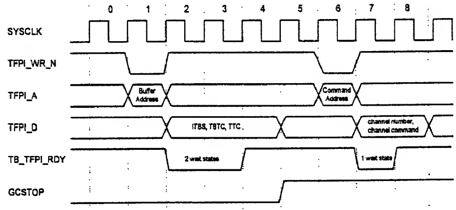


Figure 8
Configuration of a new channel

Note: TFPI interface is an 'add on'. TB macro offers a SMIF register interface with read/write signals for all readable/writeable register. As via TFPI, a transmit init/off command is executed with a write of the channel specification command register/address, a transmit debug command is excuted with a read of channel specification buffer register after a previous write of the debug command.

3.2.4 Interrupt Controller Interface (IC)

In case of not allowed combinations ITBS/TTC/TBTC a command failed (CMDF) interrupt is generated. TB delivers also a macro interrupt ID and channel number for which the 'transmit init' command failed. Another cause for CMDF is the situation, when ITBS is higher than the amount of remaining global free pool locations (i.e. all currently activated channels plus new command need buffer locations smaller than ITBS).

TB activates TB_IC_REQ_N to indicate an interrupt vector on TB_IC_D. Data are valid in cycle after IC_DTGNT

Table 6
Interrupt Vector

Bit	P1	P2	.		P3			P4	P5
Function	IID		reserved		CDMF	reserved		CHN	

IID: interrupt ID (parameter)

CMDF: command fail interrupt (bit position is parameter)

CHN: channel number

Bitpositions deifned in tb_package:
P!: tb_ic_did_lb_c
P2: tb_ic_did_rb_c
P3: tb_ic_derr_c
P4: tb_ic_dchn_lb_c
P5: tb_icd_dchn_rb_c

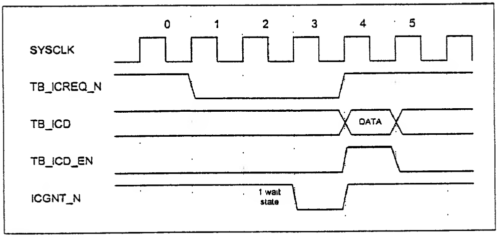


Figure 9
Interrupt Controller Interface

4 Register Description

4.1 Register Overview

Registers have to be provided to configure the channels (FIFO size, threshold value and channel number) and to read and write the internal RAMs (indirect access). and channel parameters.

4.2 Detailed Register Description

Configuration registers can be gathered togehter (TTC, TBTC, ITBS, CHN in one single register) or divided in several registers implementation dependent. Bit positions are also implementation dependent.

Following solution shows 5 register solution; with each write of the channel command register, a new configuration command is written to a configuration FIFO in TB.

Note 1: A new configuration is started with write of channel command register. To programm all channels in the same manner, only for one channel the TTC, TBTC, ... registers have to be written!

Note 2: After a successfull 'buffer create' command (i.e. buffer exists) first a 'buffer delete' command' has to be given, before the next 'buffer create' can be programmed. For details see chapter 2.

Note 3: 'P' indicates that this bit position/ bus width is a parameter.

4.2.1 Channel Command Register

Access	: write/read
Address	: *tb_vcs_cmd_adn_c*
Reset Value	: 00000000$_H$

P1 P2 P3 P4

Reserved	command	Reserved	CHN

CHN (number of bits and position are parameters):

channel number

CMD (number of bits and position are parameters):

channel command: TB provides 3 commands

- buffer create ('transmit init'),
- buffer delete ('transmit off'),
- buffer parameter debug ('transmit debug') command.

Bitpositions defined in tb_package:
P1: tb_vcs_txcom_lb_c
P2: tb_vcs_txcom_rb_c
P4: tb_vcs_chn_lb_c
P5: tb_vcs_chn_rb_c

4.2.2 Buffer Parameter Register

Access	: write/read
Address	: *tb_cvs_bufs_adn_c*
Reset Value	: 00000000$_H$

	P1 P2		P3 P4		P5 P P6
reserved	TTC	reserved	TBTC	reserved	ITBS

TBTC (number of bits and position are parameters):

Burst Threshold Code.

0: burst threshold = 1 word

others: burst threshold = $2^{(TBTC+1)}$ words (as an example, coding is also a parameter)

TTC (number of bits and position are parameters):

Transmit Threshold Code.

0: burst threshold = 1 word

others: burst threshold = $2^{(TTC+1)}$ words (as an example, coding is also a parameter

ITBS (number of bits and position are parameters):

Individual Transmit channel Buffer Size = ITBS words

Bitpositions defined in tb_package:
P1: tb_tb_vcs_ttc_lb_c
P2: tb_tb_vcs_ttc_rb_c
P3: tb_tb_vcs_btc_lb_c
P4: tb_tb_vcs_btc_rb_c
P5: tb_tb_vcs_itbs_lb_c
P6: tb_tb_vcs_itbs_rb_c

4.2.3 Indirect Access Register: Address

All rams and some register can be read/written by writing 'indirect access register address'. If autoincrement function is selected, with each read/ write access to 'indirect access register data' the next ram address is read/written. Only a startaddress has to be defined. Autoincrement may be stopped (no read/write) or interrupted (by new TYPE definition).

Access	: read/write
Address	: *tb_tac_adn_c*
Reset Value	: 00000000_H

P1 P2 P3 P4 P5

MID	reserved	AINC	reserved	COMMAND

P6 P7

Channel#/Address

MID:

Macro Idenitfication Number; access defined by TYPE etc. is only performed if MID matches the (implementation specific) ID.

AINC:

Automatic Incrementation

AINC=1 : The address will be automatically incremented with each read or write access

AINC=0 : The address won't be incremented (default value).

Bitpositions defined in tb_package:
P1: tb_vg_mid_lb_c
P2: tb_vg_mid_rb_c
P3: tb_vg_ai_c
P4: tb_vg_cmd_lb_c
P5: tb_vg_cmd_rb_c
P6: tb_vg_adr_lb_c
P7: tb_vg_adr_rb_c

COMMAND:

Bitcodings defined in tb_package:

tb_vg_cmd_fpc_c: read free pool counter with next TD read (actual value of free pool counter FPC is only readable)

tb_vg_cmd_fpp_c: read free pool pointer with next TD read (actual value of free pool pointer FPP is only readable)

tb_vg_cmd_gfpc_c: read global free pool counter with next TD read (actual value of GFPC is only readable)

tb_vg_cmd_aqctr_c: read action queue counter with next TD read (actual value of AQ read and write pointer is only readable)

tb_vg_cmd_cqctr_c: read configuration queue counter with next TD read (actual value of CQ read and write pointer is only readable)

tb_vg_cmd_pmt_on_c: switch TFPI-pmt interface on (a read of TB data is also possible via a TD read access)

tb_vg_cmd_pmt_off_c: switch TFPI-pmt interface off

tb_vg_cmd_pmt_c: in case of active TFPI-pmt interface mode the channel number is specified via Channel/ Address field). Note: In following read data via TD-read bit[31] is used as status bit, bit[30] is used to indicate TB_EMPTY condition. Therefore only a 30 bit transfer is possible for test purposes.

tb_vg_cmd_dmut_on_c: switch TFPI-dmut interface on (following read/write access to TD is a read/write of TB-DMUT-request register; note that only the CHN field is valid, i.e. only a single data word transfer is possible)

tb_vg_cmd_dmut_off_c: switch TFPI-dmut interface off

tb_vg_cmd_dmut_req_c: select dmut request register (read and write of TD is necessary: TD read delivers channel number CHN and requestad transfer length RTL, write of TD specifies only channel number, STL is ignored)

tb_vg_cmd_dmut_dat_c: select dmut data register (in this case only write of TD is possible)

tb_vg_cmd_aq_c: read/write action queue ram; Address specifies address of ram and is incremented with each read/write access of TD when AI=1 (this autoincrement feature is also active for following ram accesses) ·

tb_vg_cmd_cq_c: read/write configuration queue ram

tb_vg_cmd_pt0_c: read/write parameter table ram (lower 32 bits)

tb_vg_cmd_pt1_c: read/write parameter table ram (higher bits)

tb_vg_cmd_pt2_c: read/write parameter table ram (even higher bits)

*tb_vg_cmd_pt3_c: read/write parameter table ram (highest bits); Note: The coincidence of those tb_vg_cmds_pt*_c commands depends on implementation parameters: ITBS, number of TBTC, TTC and number of channels)*

tb_vg_cmd_ll_c: read/write link list ram

tb_vg_cmd_db_c: read/write data buffer ram

tb_vg_cmd_stat_c: read/write status bit ram

Channel#/Address:

Channel number or Address field

4.2.4 Indirect Access Register: Data

Access	: read/write
Address	: *tb_td_adn_c*
Reset Value	: 00000000_H

31	16
Test Data	

15	0
Test Data	

Test Data:

5 Appendix: M256F tb_shell

For M256F application a special naming convention is used. Addtional 'daisy chain' signals are added.

5.1 Register Description

5.1.1 Register Overview

Table 7PCI Slave Register Set (Direct Addressing)

Register Name		Read/ Write	Offset to PCI BAR1
Virtual Global Registers			
	TAC	R/W	058_H
	TD	R/W	$05C_H$
Macro Specific Registers			
Virtual Channel Specification Registers			
	CSPEC_CMD	R/W	000_H
	CSPEC_BUFFER	R/W	020_H

5.1.2 Detailed Register Description

5.1.2.1 (Virtual) Channel Specification Command (CSPEC_CMD)

Access	: write/read
Address	: 000_H
Reset Value	: 00000000_H

31	16
CMD_XMIT(7·0)	CMD_REC(&:0)

15 8	7 0
00	CHAN(7:0)

Note: The Virtual Channel Spec (VCS) Command Register has to be programmed after all other required VCS registers, in order to initiate the programming of all macros. In case of a debug command, first the command register has to be written, then a broadcast read of the virtual channelspec is possible by read of all VCS data registers. Only the macro which has implemented the corresponding bit has to drive the register bit, otherwise drives '0' to provide broadcast read feature.

CHAN(7:0): selected channel number to be programmed

CMD_XMIT(7:0): for details refer to table "Command Description"

Note: Transmit Init for a channel must be programmed only after reset or after a Transmit Off command, i.e. two Transmit Init commands for the same channel are not allowed

Command Table Transmit:

31	30	29	28	27	26	25	24	
Rsvd	Rsvd	Idle	Debug	HOLD RESET	ABORT	OFF	INIT	function
0	0	0	0	0	0	0	1	transmit init
0	0	0	0	0	0	1	0	transmit off
0	0	0	1	0	0	0	0	transmit debug
other vectors								transmit nop

5.1.2.2 (Virtual) Channel Specification Buffer (CSPEC_BUFFER)

Access	: read/write
Address	: 020_H
Reset Value	: 00200000_H

31 29	28 16
TQUEUE(2:0)	ITBS(12:0)

15 12		10 8		6 4		2 0
TTC(3.0)	0	TBTC(2.0)	0	~~RQUEUE(2.0)~~	0	~~RBTC(2.0)~~

TBTC: transmit burst threshold code
ITBS: individual transmit buffer size
TTC: Transmit Threshold Code
(for coding see chapter 5.4)

5.1.3 Test Command Register (TAC)

Access	: read/write
Offset Address	: ${}_H$
Reset Value	: 00000000$_H$

31 28				24	23 16
MID	0	0	0	AI	CMD

15			12 0
0	0	0	ADDRESS/CHANNEL

MID:	Macro ID Code
AI:	Auto Increment Function
Address:	internal address
CMD:	Command (select of ram and register)

CMD:

23	22	21	20	19	18	17	16	
								function
0	0	0	0	0	0	0	0	tb_vg_cmd_fpc_c
0	0	0	0	0	0	0	1	tb_vg_cmd_fpp_c
0	0	0	0	0	0	1	0	tb_vg_cmd_gfpc_c
0	0	0	0	0	0	1	1	tb_vg_cmd_aqctr_c
0	0	0	0	0	1	0	0	tb_vg_cmd_cqctr_c
1	0	1	0	1	1	1	0	tb_vg_cmd_pmt_on_c
1	0	1	0	1	0	1	0	tb_vg_cmd_pmt_off_c
1	0	1	0	0	0	0	0	tb_vg_cmd_pmt_c

23	22	21	20	19	18	17	16	
1	0	1	1	1	1	1	0	tb_vg_cmd_dmut_on_c
1	0	1	1	1	0	1	0	tb_vg_cmd_dmut_off_c
1	0	1	1	0	0	0	1	tb_vg_cmd_dmut_req_c
1	0	1	1	0	0	0	0	tb_vg_cmd_dmut_dat_c
1	1	1	1	0	0	0	0	tb_vg_cmd_aq_c
1	1	1	1	0	0	0	1	tb_vg_cmd_cq_c
1	1	1	1	0	0	1	0	tb_vg_cmd_pt0_c
1	1	1	1	0	0	1	1	tb_vg_cmd_pt1_off_c
1	1	1	1	0	1	0	0	tb_vg_cmd_pt2_c
1	1	1	1	1	0	0	0	tb_vg_cmd_pt3_c
1	1	1	1	0	1	0	1	tb_vg_cmd_ll_c
1	1	1	1	0	1	1	0	tb_vg_cmd_db_c
1	1	1	1	0	1	1	1	tb_vg_cmd_stat_c

Macro ID Code:

31	30	29	28	macro
0	1	0	0	TB (tb_vg_mid_tb_c)

General

- the test access provides read/write access of important internal rams and registers
- test registers are virtuals global registers (SEL signal: pb_vg_tfpi_sel_n / implemented as a daisy chain).
- the single macros are selected by the MID code of the test command register
- CMD specifies/selects one of the macro rams/registers
- the address field is used to access a ram address
- AI: autoincrement : address given in the address field is incremented automatically for each access
- All macros which are not selected by MID drive data output "00000000" and <macro>_TPFI_RDY = '1'. Driving "00000000" would mean not disable the enable line for data out, but to set the output data to "00000000".

Test write access:

1. Write TAC
2. Write TAD

Test read access:

1. Write TAC
2. Read TAD

Typically the macro selected via MID delays the RDY signal until the selected ram/register has been read and the data can be provided at the TFPI interface. No prefetch of testdata is required.

Note: CMD/Address have to be defined for each macro; there is no read/write selection in the CMD field; rd/wr's are handled with the TFPI read & write signals

5.1.4 Test Data Register (TD)

Access	: read/write
Address	: $5C_H$
Reset Value	: 00000000_H

31	
TEST DATA	

15	0
TEST DATA	

TEST DATA:	ram/register test data (read or write)

5.2 DMUT interface

Table 8
Request Register for data request from TB to DMUT

Bit	31			28	16			7	0
Function	DEL	Reserved		RTL (requested transfer length)		Reserved		CHN (Channel number)	

Table 9
Request Register for data transfer acknowledge from DMUT to TB

Bit	31			28	16			7	0
Function	CI	Reserved		STL (served transfer length)		Reserved		CHN (Channel number)	

5.3 Interrupt Controller Interface (IC)

Table 10
Interrupt Vector

Bit	31	28	.		17			7	0
Function	1001		0		CDMF	0		CHN	

5.4 Ram sizes

5.4.1 TB_PTR (TB_PTR1)

Parameter Table: 256 x 90

Table 11: All channel parameters:

ITBS	BTC	TTC	FIRST	AQE	WAIT	CIPV	CIP	FCTR	ECTR	WP	RP
(13)	(4)	(4)	(1)	(1)	(1)	(1)	(13)	(13)	(13)	(13)	(13)

5.4.2 TB_DBR and TB_LLR

Data Buffer: 8K x 33

Link List: 8K x 13

5.4.3 TB_AQR

Action Queue: 256 x 8

5.4.4 TB_CFGR (TB_PTR2)

Configuration Queue: 256 x 29

Table 12: All locations:

ITBS	TBTC	TTC	CHN
(13)	(4)	(4)	(8)

5.5 Gate count

M256F implementation needs 21.3k gates (1360 flipflops).

5.6 Shell Interface and Signal Description

M256F Symbol name	I/O	Macro Symbol Name	Function

Clock and Reset

SYSCLK	I	SYSCLK	Internal system clock (33 MHz)
HW_RESET_N	I	(RESET_N)	General hardware reset of TB. All registers and RAM reset or initialization.
SW_RESET_N	I	-	General sofrware reset of TB. All registers and RAM reset or initialization.
SCANMODE	I	-	Scanmode
GC_STOP	I	GCSTOP	Stop all non configuration process in TB (fast programm mode)
TB_IIP	O	TB_IIP	Initialization of TB rams in progress

Protocol Machine Receive (PMT) Interface

PT_TB_RD_N	I	PTRD	PMT read. Only single word write transfer is supported.
PT_TB_A [CNB-1:0]	I	PTA	Address bus. Specifies channel number for transfer.
TB_PT_D [DBB-1:0]	O	TB_PTD	TB Data/Status word being transferred to PMT.
TB_PT_RDY	O	TB_PTRDY	Ready. End of data transfer indication. 0 => TB inserts wait state. 1 => TB will finish transfer during next clock cycle.
TB_PT_STAT	O	TB_PTSTAT	Mode of data word to be transferred. 0 => protocol data 1 => status + protocol data

M256F Symbol name	I/O	Macro Symbol Name	Function
TB_PT_EMPTY	O	TB_PTEMPTY	Asserted for one cycle while PMT read if TB has no data stored for the channel specified by the address bus.

DMU Transmit (DMUT) Interface

DT_TB_RD_N	I	DTRD	Read. DMUT Read Control (for request register)
DT_TB_WR_N	I	DTWR	Write. DMUT Write Control (for request register or data register)
DT_TB_A	I	DTA	Address bus (1 bit) 0 => Request register. 1 => Data register.
DT_TB_D[DBB-1:0]	I	DTD	Data in. Input command from DMUT (channel number e. g.) or data/ status input
DT_TB_STAT	I	DTSTAT	Status indication from DMUT
TB_DT_REQ_N	O	TB_DTREQ	Service request from TB to DMUT controller. Asserted as long as TBAQ is not empty.
TB_DT_D[DBB-1:0]	O	TB_DTD	Data out. Request register from TB to DMUT controller
TB_DT_RDY	O	TB_DTRDY	Ready. End of data or command transfer indication. 0 => TB inserts wait state. 1 => TB will finish transfer during this clock cycle.

Interrupt Controller (IC) Interface

M256F Symbol name	I/O	Macro Symbol Name	Function
TB_IC_REQ_N	O	TB_ICREQ	Request Line
IC_TB_GNT_N	I	ICTBGNT	Grant Line
TB_IC_D[31:0]	O	TB_ICD	TB interrupt vector data
IC_D	I	-	daisy chain data input

FPI Slave Interface

PB_TFPI_RD_N	I	TFPI_RD_N	TFPI read TB
PB_TFPI_WR_N	I	TFPI_WR_N	TFPI write to TB
PB_TFPI_A[8:2]	I	TFPI_A[TFPIAB-1:2]	Address bus.
PB_TFPI_D[31:0]	I	TFPI_D	Input Data. Active during data phase of read cycle.
PB_TFPI_RDY	I	TFPI_RDY	TFPI ready input
PB_VC_TFPI_SEL_N	I	TFPI_VC_SEL_N	Slave select.
PB_VG_TFPI_SEL_N	I	TFPI_VG_SEL_N	Slave select.
TFPI_D[31:0]	O	-	Daisy chain input.
TB_TFPI_RDY	O	TB_TFPI_RDY	End of transfer indicator: 0 => Master should insert wait states 1 => TB will complete transfer in this cycle
TB_TFPI_D[31:0]	O	TB_TFPI_D	Output Data. Active during data phase of write cycle.

5.7 Dimension of M256F application and Package definitions:

Name	DescriptionO	M256F value
tb_chn_nw_c	number of supported channels	256
tb_chn_nb_c	channel bus bits	8
tb_db_nw_c	depth of data buffer	8192
tb_dba_nb_c	data buffer address bus bits	13
tb_btc_nb_c	number of TBTC bits	4

tb_ttc_nb_c	number of TTC bits	4
tb_btc2btl	coding of TTC values	0: 1 1: 4 2: 8 3: 12 4: 16 5: 24 6: 32 7: 40 8: 48 others: 64
tb_ttc2ttl	coding of TTC values	0: 1 1: 4 2: 8 3: 12 4: 16 5: 24 6: 32 7: 40 8: 48 9: 64 A: 96 B: 128 C: 192 D: 256 E: 384 F: 512
tb_data_nb_c	data bus width	32
Initialisation of rams		
tb_iip_pt1_en_c	to reduce power consumption during initialisation (TB_IIP = '1'), this parameter shifts PT1 ram initialisation to later times. E.g. Time of M256F inittime is defined by data buffer size of 8192 dwords. IIP phase of PT1 ram starts at cycle bin:1110100000000 (hex: 1D00) when tb_iip_pt1_en_c is set to '11101'	11101
tb_iip_aq_en_c	Shift of AQ ram init phase (start init at hex:1E00)	11110

tb_iip_cq_en_c	Shift of CQ (configuration queue) init phase (start init at hex:1F00)	11111
PMT interface		
tb_pt_rd_act_c	active level of PMT read signal	0
DMUT interface		
tb_dt_wr_act_c	active level of DT_WR write signal	0
tb_dt_rdy_act_c	active level of TB_DTRDY ready signal	1
tb_dt_req_act_c	active level of TB_DTREQ request signal	0
tb_dt_a_command_c	address of command/request register	0
tb_dt_crbl_rb_c	bit position of RTL in request register	16
tb_dt_crci_pb_c	bit position of CI (complete indication) in command register	31
tb_dt_crdel_pb_c	bit position of DEL (delete) in command register	31
tb_dt_crchn_rb_c	bit position of CHN (channel number) in command register	0
TFPI interface		
tb_sfr_addr_c	number of via TFPI addressable registers	4
tb_fpi_rd_act_c	active level of TFPI read	0
tb_tpi_wr_act_c	active level of TFPI write	0
tb_fpi_rdy_act_c	active level of TFPI ready	1
tb_fpi_sel_act_c	active level of TFPI selects (vg/vc)	0
tb_fpi_en_act_c	active level of TFPI output enable	1
tb_fpi_a_rb_c	TFPI address bus (LSB)	2
tb_fpi_a_lb_c	TFPI address bus (MSB)	8
tb_vcs_cmd_adn_c	channel command address	00(hex)
tb_vcs_txcom_rb_c	bitposition of command in channel command	24
tb_vcs_txcom_lb_c		31
tb_vcs_tx_initv_c	transmit init command vector	01
tb_vcs_tx_offv_c	transmit off command vector	02
tb_vcs_tx_debugv_c	transmit debug command vector	10
tb_vcs_chn_rb_c	bitposition of channel number in channel command	0

tb_vcs_bufs_adn_c	buffer register address	20
tb_vcs_itbs_rb_c	bitposition of ITBS in buffer register (LSB)	16
tb_vcs_btc_rb_c	bitposition of TBTC in buffer register (LSB)	4
tb_vcs_ttc_rb_c	bitposition of TTC in buffer register (LSB)	8
tb_tac_adn_c	test access command register address	58
tb_vg_mid_rb_c	bitposition of MID (macro ID) in TAC register (LSB)	28
tb_vg_mid_lb_c	bitposition of MID (macro ID) in TAC register (MSB)	31
tb_vg_mid_tb_c	MID of TB	4
tb_vg_ai_c	bitpostion of AI (auto increment bit) in TAC	24
tb_vg_cmd_rb_c	bitposition of CMD in TAC register (LSB)	16
tb_vg_cmd_lb_c	bitposition of CMD in TAC register (MSB)	23
tb_vg_adr_lb_c	bitposition of ADR/CHN in TAC (LSB)	0
tb_vg_adr_rb_c	bitposition of ADR/CHN in TAC (MSB)	12
tb_td_adn_c	test data register address	5C
tb_vg_cmd_fpc_c	test command for intternal register	00
tb_vg_cmd_fpp_c		01
tb_vg_cmd_gfpc_c		02
tb_vg_cmd_aqctr_c		03
tb_vg_cmd_cqctr_c		04
tb_vg_cmd_pmt_on_c	test command for PMT interface mode	AE
tb_vg_cmd_pmt_off_c		AA
tb_vg_cmd_pmt_c		A0
tb_vg_cmd_dmut_on_c	test command for DMUT interface mode	BE
tb_vg_cmd_dmut_off_c		BA
tb_vg_cmd_dmut_req_c		B0
tb_vg_cmd_dmut_dat_c		B1
Test dump/write feature of rams and register		
tb_test_slice_nb_c	number of testslices for broad rams	4
tb_vg_cmd_aq_c	test command for rams	F0
tb_vg_cmd_pt0_c		
tb_vg_cmd_pt1_c		

tb_vg_cmd_pt2_c		
tb_vg_cmd_pt3_c		
tb_vg_cmd_ll_c		
tb_vg_cmd_db_c		
tb_vg_cmd_stat_c		
tb_test_pt*_**_c (*: 1,2,3 and **: rb,lb)	selected slices for testmode access of parameter table ram	1: lb=63/rb=32 2: lb=89/rb=64 3: lb=89/rb=64 (3 not used)
IC interface		
tb_ic_req_act_c	active level for TB_ICREQ	0
tb_ic_gnt_act_c	active level for IC_GNT	0
tb_ic_dchn_rb_c	bitposition of channel number in interrupt vector (LSB)	0
tb_ic_did_nb_c	bitposition of interrupt ID in interrupt vector (number of bits)	4
tb_ic_did_rb_c	(LSB)	28
tb_ic_did_code_c	ID code	1001
RAM interface		
tb_wnram_act_c	active level of ram write signal	0
tb_bsnram_act_c	active level of ram select	0
Global interface		
tb_iip_act_c	active level of IIP (initialisation in progress) signal	1
tb_reset_act_c	active level of reset signal	0
tb_stop_act_c	active level of GC_STOP signal	1

Appendix I

Title:
Receive Buffer V2.1

1 Introduction

1.1 Overview

The Receive Buffer (RB) provides buffering of frame data and status between a Protocol Machine Receive (PMR) and Data Manage Unit Receive (DMUR). The PMR transfers data or status in 32 bit dword units to the RB which accumulates them in per channel buffers. When the count of dwords reaches a programmed burst size or a status word is transferred (status indicates end of frame or error), the RB transfers that channel's data to the DMUR as a block. The DMUR can then transfer data on the PCI bus in burst mode which results in more efficient bus usage.

The number of channels supported by the RB and the size of data buffers and burst event queues is fixed at synthesis time by VHDL package parameters. Buffers are built dynamically during operation from a free pool which should be dimensioned for the application dependent on number of channels and burst size. The throughput of the RB is naturally dependent on the speed of the PMR and DMUR but is ultimately limited by the RB's memory architecture (single port RAM and shared buffer pool). For long frames and large burst size, throughput could approach 3 clocks per double word but a more realistic limit for the average application is 4 clocks per dword.

The maximum aggregate serial bit rate is a function of system clock and the protocol frame size. Assuming a fully transparent protocol (no flags, bit stuffing), a 33MHz clock, and a conservative 5 clocks per dword results in 32 x 33/5 = 211 Mbit/second aggregate.

Features including performance, power usage, number of gates, area

- Buffers frame data and status per channel from a free pool common to all channels
- Per channel burst trigger. User specified (1, 4, 8, 16, 32, or 64 dwords) threshold
- Built In Self Test of RAM (BIST)
- system clock up to 66 MHz
- number of gates:
- area:
- power consumption:
- scan path

1.2 System Integration and Application

The RB interfaces are:

- configuration and control (Simplified Microprocessor Interface or SMIF)
  register oriented with separate read/write strobes per register

- Protocol Machine Receive (PMR)
  dword transfer. transfers blocked by flag when RB buffers flag.

- Data Management Unit Receive (DMUR)
  block oriented transfer of up to 64 dwords.

- Interrupts for buffer supervision

RB provides supervision of the free buffers in the data pool and action queue pool. RB interrupts can be asserted when buffer pool counts fall below user programmable thresholds. This provides an overload indication for flow control mechanisms and/or traffic engineering.

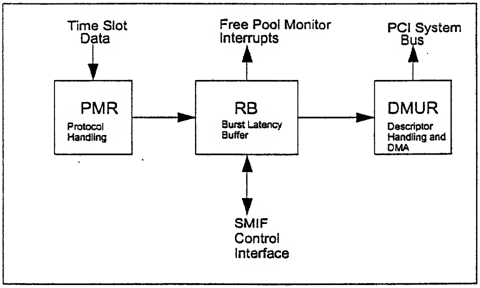


Figure 1
System Integration

1.3 Known Restrictions and Problems

2 Functional and Test Description

2.1 Block Diagram incl. Clocking Regions

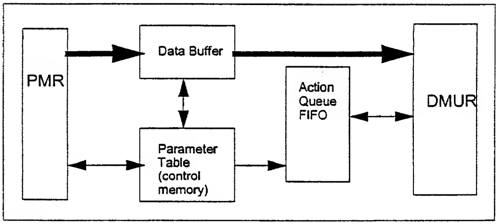


Figure 2
RB Block Diagram

2.2 Normal Operation Description

As shown in **Figure 2**, the main RB functions are realized in 3 blocks:

- Data Buffer (RBDB) for data buffers
- Parameter Table (RBPT) for control of individual channel queues
- Action Queue (RBAQ) for buffering of block requests for the DMUR

The PMR transfers data to the RB when:

- a 32 bit data word is filled with service data (data word) or
- end of frame or error condition is recognized (status word). a status word may contain from 0 to 3 octets of service data in addition to 1 octet of status.

PMR transfers a single dword by asserting a write strobe concurrently with:

1. data/status word (32 bit)
2. channel address (application dependent width)
3. control flag indicating the dword type (either service data or status+service data),
4. count of service data octets (only used if status word)
5. flag indicating if data currently stored in the RB for the channel should be discarded.

The channel number forms an index into the RB's parameter table which contains the channel's buffer control data. An element from a free buffer pool in RBDB is written with the new dword and linked to the channel's buffer chain. The free pool counts in RBDB and the read/write pointers and word count values in RBPT are updated.

When the channel's programmed burst threshold is reached or a status word is being transferred (end of frame or error), an entry is written to the RB Action Queue (RBAQ). A FIFO entry identifies the channel number, count of words to be transferred, type of the last dword in the burst (data or status+data), and the count of service data octets in a status word. Note that only the last word of a block being transferred is allowed to be status. The data and/or status dwords remain in the RBDB until transferred to the DMUR channel. When the FIFO is not empty, RB asserts a service request to the DMUR controller which initiates the data transfer.

If either RBDB or RBAQ becomes full, RB generates a buffer overflow indication by setting the RB_PRFULL line high. PMR must suppress any new data transfers. PMR is responsible to report the overflow.

A buffer discard function is activated by asserting PRDISCARD during the transfer. This allows PMR to discard channel data being accumulated in RB but which has not yet been transferred to the action queue. This feature is useful when the Protocol Machine recognizes that a frame is errored (e.g. too short) and should not be transferred to the software input queue. A discard for a channel should only be requested if the number of blocks transferred to the RB since the last status word is less than the channel's burst length. RB does not verify this.

2.2.1 Receive Buffer Parameter Table (RBPT)

The RBPT RAM provides a control word per channel for buffer management:

- Channel burst threshold
- Count of dwords in buffer chain not already sent to action queue FIFO
- Pointers for start and end of buffer chain (RBDB read and write addresses)
- Channel buffer status (empty/not empty)

Empty flag	Pre-Burst Count PBCNT (PC)	RD Pointer (BA)	WR Pointer (BA)	BTC (BC)

Burst Threshold Codes (BTC) are application dependent and fixed at synthesis by a table of constants in a VHDL package. Typical application are expected to use between 8 and 16 codes but this is not fixed in the architecture.

The width of the address bus BA is dependent on number of words RBDBS in the RAM.

$2^{BA-1} < RBDBS <= 2^{BA}$

When the accumulated count of dwords reaches the burst threshold or a status/discrd word is received, the chain of dwords is transferred to the Action Queue FIFO. The Pre-Burst Count (PBCNT) is the count-1 of words in the chain before it is transferred to the FIFO.

2^{PC} = maximum supported burst length e.g. 6 bits will support 64 word burst length.

The parameter table is implemented in 2 separately addressable RAMs to allow the SMIF interface to update the BTC without waiting on PMR/DMUR access of the PT:

1. Burst Parameter Code (BC)

MaxNumChan rows X BC columns

2. Channel Buffer State (read and write pointers, pre-burst count, and buffer empty flag)

. MaxNumChan rows X (2BA+PC+1) columns

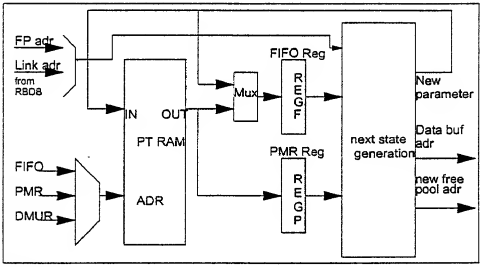


Figure 3
Receive Buffer Parameter Table (RBPT)

The pre-burst count PBCNT only has to count to 2^{PC}-1 for burst size 2^{PC}. The next dword transferred from PMR will generate an action request for the RBAQ FIFO and reset the PBCNT field to zero. The buffer empty flag is needed and cannot be derived from the PBCNT or read/write pointers because

1. PBCNT is zero following the transfer to the FIFO but dwords remain in the chain until the RBAQ FIFO has been serviced and

2. PBCNT=0 and WP=RP when burst length=1 or a single status word is transferred.

The RBPT is accessed by both the PMR and the DMUR service. PMR and DMUR (FIFO) service routines use separate output registers (REGR and REGF respectively) for the parameter table.

PMR service has the highest priority because it is essentially non-deferable.

PMR service when adding data to a buffer chain

A PMR request writes a single entry to RBDB. A read-modify-write of the channel's RBPT entry is performed using REGP. This can never be delayed by FIFO or slave FPI bus access.

FIFO service (RBAQ) when removing data from a buffer chain

A FIFO entry typically will read the parameter table, release multiple entries in the RBDB depending of burst length, and then write back the parameter table. The parameter table entry is copied to the REGF output register and updated during the burst. The RBPT ram is only updated at the end of the burst when it writes back REGF.

During the time a channel's RBPT entry is stored in the REGF, PMR service can access it to write data and update the channel's parameter table entry without delay. FIFO service is delayed during PMR service if this access is required. The PMR can not be deferred

2.2.2 RB Data Buffer (RBDB)

The RBDB Size (RBDBS) is sized according to:

- number of channels supported
- sum of channel bit rates
- burst threshold on bus
- maximum bus latency

An entry consists of a link field (BA bits wide) and data field (DB bits wide). The data and link fields are implemented in 2 physical RAMs to allow simultaneous read and write access of the data and link pointer fields at the same address.

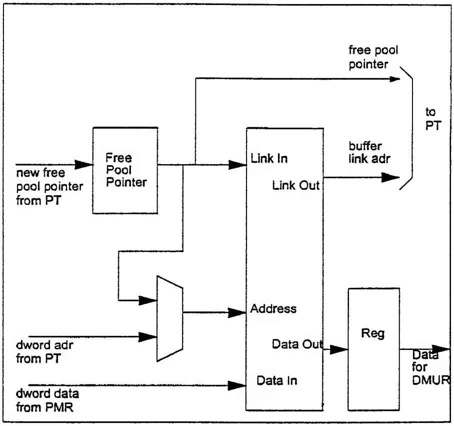


Figure 4
Structure of RB Data Buffer (RBDB)

2.2.3 RB Action Queue (RBAQ)

RBAQ is the FIFO which buffers the requests from RB to the DMUR Receive channel. An entry specifies:

- Channel Number
- Burst length
- Data type of last dword (protocol data or status)

- Number of service data octets in status dword

The data associated with a FIFO event remains in the RBDB until read by the DMUR. Because data can be delayed due to the higher priority PMR service, a ready signal is required at the DMUR interface for data transfers.

RBAQ size (RBAQS) is dimensioned according to:

- sum of channel bit rates
- minimum burst threshold on bus (1 in worst case)
- maximum bus latency

RBAQS words x (B+C) bits

S(1)	OC	BurstLength (BL)	Channel # (CN)

C depends on Maximum Number of Channels (MNC) supported in applicatio
2^{C-1} < MNC <= 2^{C}
BurstLength is the number of words-1 to be transferred.

BL depends on maximum Burst Length supported in the device.
External Burst Length = Burst Length+1 =<2^{BL}
S flag type of last dword in burst:
0 => protocol data word
1 => status word

Octet count (OC) field specifies number of service data octets in status word

Figure 5
Layout of the RB Action Queue (RBAQ)

The discard of data chains which occurs PMR asserts PRDISCARD during a dword transfer is handled during normal FIFO event processing. The FIFO entry is coded as a non-status event (S=0) but with octet count OC/= 0. Normal data entries write OC=0. When the event is read, the buffer segment is walked but no DMUR bus cycle is executed. Dummy DMUR bus cycles are generated internally so that the sequencers run identically whether a normal DMUR access or discard cycle is running. During this time, the RB does not assert any service request toward the DMUR so that no conflicts will arise between the real and dummy DMUR.

2.2.3.1 Initialization

At reset, all RAMs are automatically initialized. The RB_IIP (Initialization in Progress) flags is asserted active until this is completed and the RB is ready for software initialization. The RBDB data RAM is written with all zeros while the links are chained into the Free Pool (FP). The Free Pool Pointer (FPP) is a register pointing to the start of the chain. A channel buffer is a linked list built from elements taken from this pool.

All RBPT entries are initialized to empty and burst threshold code equal to a default reset value specified in a VHDL package.

The RBAQ (FIFO RAM) is cleared for testing purposes. The FIFO read and write pointer registers are initialized to 0 and the FIFO status set to empty.

2.2.3.2 FIFO service Release of Buffers to Free Pool

FIFO service does not release individual dwords to the RBDB free pool as they are transferred to DMUR. Instead it waits until the last dword of the burst is read. The resulting free buffer chain or segment is released in a single clock by copying the segment start address to the FPP and the old FPP to the link field of the dword at the segment end address. Two auxiliary registers are required to store the beginning and end addresses of the chain being transferred to the DMUR interface:

SEGS = RP (start address of buffer chain segment begin read)

SEGE = RP (end address of buffer chain segment begin read)

2.3 Reset Behavior

2.4 Functional Test Description

2.5 Production Test Description

3 Interfaces and Signal Description

A signal is active high unless "_N" is appended to its name. To make the design as re-usable as possible, a bus signal whose width is application dependent will be specified by one of the following parameters:

Parameter name	Bus Type	Typical value (bits)
CN	Channel Number	8 (256 channels)
DB	Data/Status	32
BT	Burst Threshold Code	4 (max 16 codes)

BL	Burst Length	6 (max burst 64 dw)
OC	Status Service Data Octet Count	2 (0-3 octets)
BA	RB Data Buffer Address	12 (max 4095 dwords)
AQA	Action Queue RAM Address	9 (0 to 512 entries)

3.1 Signal Description

Table 1
Macro Interfaces and Signal Description

Symbol name	I/O	Function
Clock and Reset		
CLK	I	Internal clock (66 MHz)
RESET_N	I	General reset of RB. All registers and RAM reset
STOP	I	Test mode (active high). Stops all channel sequencers.
RB_IIP	O	RB initialization in progress following reset. RB is not available when this signal is asserted. This signal will remain high for 3072 clocks following release of RB reset(s).
Protocol Machine Receive (PMR) Interface		
PRWR_N	I	**Operation Code.** PMR write cycle. Single dword.
PRA[CN-1:0]	I	**Address bus.** Specifies channel number for transfer.
PRD [DB-1:0]	I	**PMR Data/Status** dword being transferred
PRSTAT	I	Mode of data word to be transferred. 0 => protocol data 1 => status + protocol data (0 to 3 octets)
PRSTAT_OC [OC-1:0]	I	Count of data octets in status word being transferred
PRDISCARD	I	Discard open data block for the channel not yet transferred to FIFO.
RB_PRRDY	O	**Ready** End of data transfer indication. 0 => PMR should insert wait state. 1 => RB will finish transfer during this clock cycle.
RB_PRFULL	O	Asserted at end of PMR write cycle (RB_PRRDY=1) to indicate that no additional PMR requests can be accepted due to: • Empty data buffer free pool or • Full FIFO task buffer to the DMUR. De-asserted after DMUR restores free buffers.
Data Management Unit Receive (DMUR) Interface		

Table 1
Macro Interfaces and Signal Description (cont'd)

Symbol name	I/O	Function
DRRD_N	I	DMUR FPI read command. Only single word read transfer is supported but data and address cycles can be overlapped.
DRA	I	1 bit address 0 => status port (channel address, dword count) 1 => data port
RB_DRD[DB-1:0]	O	Output data/status from RB to DMA controller
RB_DRRDY	O	End of data transfer indication. 0 => DMUR should insert wait state. 1 => RB will finish transfer during this clock cycle.
RB_DRSTAT	O	Last word in burst is status
RB_DRREQ_N	O	Service request from RB to DMUR controller. Asserted when RBAQ is not empty.

SMIF Control Interface

BPI_RD_SFR_N[6:1]	I	Read special function registers (select per register) 1 : channel burst length parameter 2 : free pool monitoring mode and interrupt flags 3 : trigger threshold triggers for free pool monitoring 4 : test access command register 5 : test access data register 6 : count of free pool elements (data buffer and fifo)
BPI_WR_SFR_N[5:0]	I	Write special function registers (select per register) 0 : channel command register 1 : channel burst length parameter 2 : free pool monitoring mode and interrupt flags 3 : trigger threshold triggers for free pool monitoring 4 : test access command register 5 : test access data register
BPI_REQ_N	I	Special function register read or write is valid
BPI_DI[DB-1:0]	I	Data input bus
BPI_RDY_N	O	Data cycle ending

Table 1
Macro Interfaces and Signal Description (cont'd)

Symbol name	I/O	Function
BPI_D_O[DB-1:0]	O	Data output bus
Interrupt Interface (IC)		
RB_DBFP_INT	O	RB data buffer free pool threshold interrupt
RB_AQFP_INT	O	RB Action queue free pool threshold interrupt
INT_ACK	I	Interrupt acknowledged
Data Buffer Data/Status RAM Interface (M256F)		
RB_DSR_WN	O	write strobe (active low)
RB_DSR_BSN	O	block select (active low)
RB_DSR_ADR(BA-1:0)	O	address bus
RB_DSR_DI(31:0)	O	write data
RBR_DSR_DO(31:0)	I	read data (output of RB Data Buffer RAM)
Data Buffer Link RAM Interface (M256F)		
RB_DLR_WN	O	write strobe (active low)
RB_DLR_BSN	O	block select (active low)
RB_DLR_ADR(BA-1:0)	O	address bus
RB_DLR_DI(16:0)	O	write data
RBR_DLR_DO(16:0)	I	read data (output of RB Data Link RAM)
Burst Parameter RAM Interface (M256F)		
RB_PTR0_WN	O	write strobe (active low)
RB_PTR0_BSN	O	block select (active low)
RB_PTR0_ADR(CN-1:0)	O	address bus
RB_PTR0_DI(BT-1:0)	O	write data
RBR_PTR0_DO(BT-1:0)	I	read data

Table 1
Macro Interfaces and Signal Description (cont'd)

Symbol name	I/O	Function
Parameter Table RAM Interface (M256F) Parameter Table stores Data buffer read and write pointers, burst threshold code, pre-burst count, and empty flag i.e. PTD=2*BA+BT+BL+1		
RB_PTR1_WN	O	write strobe (active low)
RB_PTR1_BSN	O	block select (active low)
RB_PTR1_ADR(CN-1:0)	O	address bus
RB_PTR1_DI(PTD-1:0)	O	write data
RBR_PTR1_DO(PTD-1:0)	I	read data
Action Queue RAM Interface (M256F) Action Queue Data Bus dimensioned according to channel number, burst length, status flag, and status octet count i.e. AQD = CN+BL+1+OC		
RB_AQR_WN	O	write strobe (active low)
RB_AQR_BSN	O	block select (active low)
RB_AQR_ADR(AQA-1:0)	O	address bus
RB_AQR_DI(AQD-1:0)	O	write data
RBR_AQR_DO(AQD-1:0)	I	read data

3.1.1 RB Interface to the Protocol Machine Receive (PMR)

The PMR initiates data transfer with an address cycle i.e. active write signal PRWR_N and all control data valid (channel address, status flag, octet count, and discard flag). RB will read the PMR's data with 1 wait state and assert RB_PRRDY. PMR can write 1 dword every 3 clock cycles.

The out-of-band signals are:

- PRSTAT to indicate if transferred word is status instead of pure 32-bit data. Captured by RB during address phase.
- PRSTAT_OC to indicate the number of service data octets are contained in the status word when PR_STAT is asserted.
- PRDISCARD to indicate that this dword and any other dwords not transferred to the DMUR FIFO should be discarded.

- RB_PRFULL to stop PMR transfers when the RB has no free buffer space.

The PMR interface does not support overlapped transfers.

The RB_PRFULL signal will be asserted simultaneously with RB_PRRDY. **PMR must not attempt another transfer until this flag is de-asserted.**

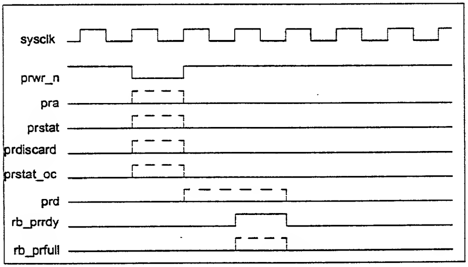


Figure 6
FPI Dword Transfer from PMR to RB

3.1.2 RB Interface to Data Management Unit Receive (DMUR)

The RB interface to the DMUR is similar to a unidirectional FPI slave bus timing. Because of the simple nature of the interface, opcode and select signals are not necessary. DMUR reads a status register when DRA=0 and the FIFO data port when DRA=1.

Two out-of-band signals are required:

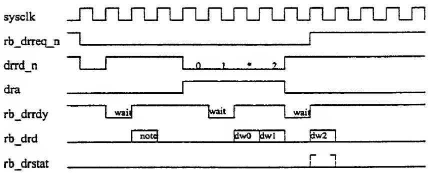
1. RB_DRREQ_N to indicate that the FIFO is not empty.
2. RB_DRSTAT to indicate if dword being transferred is status or only data. This signal is asserted only during the last dword of a burst when that dword contains status. The last dword is the only allowed position for a status word in a burst.

DMUR initiates a block transfer by asserting a read request at address 0. During the data phase, RB asserts RB_DRRDY along with the status register shown in Table 2.

Table 2
RB DMUR Status Register

Bit	31...27	26 ...25	24	23..22	21..16	15..8	7..0
		Octet count in status dword	Status flag		Burst length		Ch #

DMUR then transfers "Burst length" + 1 dwords with either individual or overlapped read cycles at word address 1. Transfers are synchronized with the ready signal. Internally the PMR has higher priority and will force additional wait states in the event of conflict. Note that although the DMUR could overlap the first data transfer cycle with the status register read, this may not be useful in practical designs as the DMUR would not know the required burst read length. The current implementation of RB always asserts at least 1 wait state at the beginning of each DMUR read opeation and at least 1 wait state after every 2 words in a burst. Additional wait states are inserted due to PMR access contention (approximately 2 clocks per PMR write access will create contention for longer DMUR burst access N > 4). For single DMUR data transfers, contention increases. There is never contention for status register read.



note: status register read returns channel number, status flag, and dword count (N).
in this example N=2 i.e.3 dword transfer.

* read 1 repeated because ready not asserted.

Figure 7
DMUR transfers 3 dwords

3.1.3 RB Control Bus (SMIF)

The SMIF interface provides a set of registers for configuration and testing. Register layouts are specified by package constants for an application and are therefore only funcitonally described here. Refer to the Appendix for an application example.

Addressing of channel specific data and all intemal RAM is done using an indirect address register.

SFR R/W index		Description
0	W	Channel Command Register Channel Number: CN bits specify the channel number for the command. Commands (at least 2 bits): The following commands are supported: 1. initialize channel 2. debug (writes channel number only) 3. no operation
1	R/W	Burst Threshold Code BT bits specify a burst code which translates to a burst length in dwords. All definitions of the burst length are specified in the VHDL package.

SFR R/W index		Description
2	R/W	Configuration Register Two flags are provided to configure the Free Pool (FP) Monitor. The bit positions of these flags are specified in the VHDL package. a. Free Pool (FP) Monitor Mode 0 -> capture minimum FP count 1 -> interrupt if FP count falls below threshold programmed in Threshold Register. b. RB FP Interrupt Mask 0 -> enable FP monitor interrupt 1 -> disable FP monitor interrupt
3	R/W	Free Pool Interrupt Threshold Parameters Bits Definition 0 - BA-1 RBDB free pool threshold 16 - 15+AQA RBAQ free pool threshold If free pool mode flag is '0' (minimum pool capture mode), reading this register returns the minimum free pool counts detected since the last read of this register. The read access resets the counts to maximum. If free pool mode is '1' (threshold interrupt mode), reading this register returns the current programmed threshold values.

SFR R/W index		Description
4	R/W	Test Command Register RAM address (BA bits) : RAM address used by access to Test Data Register. Placement of this bit field in the SMIF data bus is set in VHDL package. Its width is assumed to be BA bits which is the largest address range in the RB. Test Commands (at least 3 bits): A command opcode supports RAM testing and set/reset of test mode which forces a buffer full state. Width and placement of this opcode bit field is specified in VHDL package. a. R/W Burst Threshold Parameter RAM b. R/W Parameter Table (channel state) c. R/W Data Buffer RAM d. R/W Link RAM e. R/W Action Queue RAM f. Set test mode g. Reset test mode auto increment mode (1 bit) : following read or write of Test Data Register 0-> no post increment of address register 1-> post increment of address register command validation code : RB accepts command only if code matches constant specified in VHDL package. Width specified by application.

SFR R/W index		Description
5	R/W	Test Data Register. Read or write of this register reads or writes RAM specified in Test Command Register. If auto increment mode is active, the address will be automatically advanced to next word address. The address will wrap around automatically at the end of the selected RAM. Width of data bus depends on addressed RAM. A data is aligned to bit 0 of the SMIF data bus.
6	R	Current Free Pool Count (read only) 0 - BA-1 Data Buffer FP count 16-15+AQA FIFO FP count

RB Application : M256F

A.1 Introduction

The M256F Network Controller uses the RB Macro as a buffer between its protocol machine Receive (PMR) and Data Management Unit Receive (DMUR). The application wraps the RB in a shell for:

- Adaptation of SMIF registers to Slave FPI Bus with daisy chained data bus
- Mapping of RB interrupts to daisy chained interrupt vector bus
- Hardware and Software controlled reset capability

The M256F dimensioning parameters which are specified at synthesis time are:

- 256 channels
- 3072 dword data buffer
- 512 word FIFO (action queue for DMUR read requests)
- 10 burst codes (1, 4, 8, 12, 16, 24, 32, 40, 48, 64)
- Macro ID for test access = "0011"
- 8-bit channel command field: nop="00000000", init="00000001",debug="00010000"

The RB shell structure is shown in Figure 1.

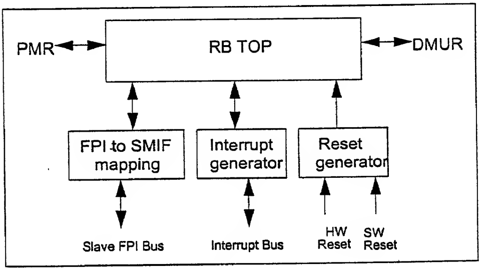


Figure A.1

RB Shell Structure for M256F Application

A.2 RB Dimensioning

A.2.1 Receive Buffer Parameter Table (RBPT)

The RBPT RAM control word for the M256F stores 12 bit addresses for the data buffer (3K words) and supports 10 burst codes (4 bits) with a maximum burst length of 64 (6 bits).

The parameter table is implemented in 2 separately addressable RAMs :

1. 256 x 4 bits for Burst Parameter Code (BC)
2. 256 x 31 bits for (read and write pointers, pre-burst count, and buffer empty flag)

30	29					24	23											12	11											0
E	burst count						Data Buffer Read Pointer												Data Buffer Write Pointer											

Figure A.2 Bit Assignments in Parameter Table 1 RAM (all but Burst Code)

The E bit is the empty flag. When E=1, the channel has an empty buffer chain and no FIFO Action Queue events.

A.2.2 RB Data Buffer (RBDB)

The M256F data buffer is implemented in 2 physical RAMs:

1. 3072 x 32 bits (dword buffer)
2. 3072 x 12 bits (chain link buffer)

A.2.3 RB Action Queue (RBAQ)

The Action Queue is implemented in a 512 word x 17 bit RAM. The status flag S and Octet Count OC are fixed for all 32 bit applications. The Burst Length and channel number are application specific.

16																0
S	OC		Burst Length						Channel Number							

Figure A.3 Bit Assignments in Action Queue RAM

Note that if S=0 and OC /= 0, the event is a discard event i.e. RB will read the data buffer chain of "Burst Length" dwords but not report them to the DMUR.

A.3 M256F RB Shell Signals

Table A.3
RB Interfaces and Signal Descriptions for M256F

Symbol name	I/O	Function
Clock and Reset		
SYSCLK	I	Internal clock (66 MHz maximum)
HW_RESET_N	I	General reset of RB. All registers and RAM reset Unclocked assert, de-assert synchronous to SYSCLK
SW_RESET_N	I	General reset of RB. All registers and RAM reset Assert and de-assert synchronous to SYSCLK
GC_STOP	I	Test mode (active high). Stops all channel sequencers.
RB_IIP	O	RB initialization in progress following reset. RB is not available when this signal is asserted. This signal will remain high for 3072 clocks following release of RB reset(s).
Protocol Machine Receive (PMR) Interface		
PR_RB_WR_N	I	**Operation Code.** PMR write cycle. Single dword.
PR_RB_A[7:0]	I	**Address bus.** Specifies channel number for transfer.
PR_RB_D [31:0]	I	**PMR Data/Status** dword being transferred
RB_PR_RDY	O	**Ready** End of data transfer indication. 0 => PMR should insert wait state. 1 => RB will finish transfer during this clock cycle.
PR_RB_STAT	I	Mode of data word to be transferred. 0 => protocol data 1 => status + protocol data
PR_RB_DISCARD	I	Discard data not yet written to Action Queue. only evaluated if PR_RB_STAT=1. 0 => no discard 1 => discard
PR_RB_STAT_OC [1:0]	I	Count of data octets in status word being transferred

Table A.3
RB Interfaces and Signal Descriptions for M256F

Symbol name	I/O	Function
RB_PR_FULL	O	Asserted at end of PMR write cycle if: • Buffer free pool becomes empty or • FIFO task buffer to the DMA channel becomes full. De-asserted after DMA read cycle.
Data Management Unit Receive (DMUR) Interface		
DR_RB_RD_N	I	DMUR FPI read command. Only single word read transfer is supported but can be overlapped.
DR_RB_A	I	select status or data port 0 => read status register (block size, channel number) 1 => data port (data or status dword)
RB_DRD[31:0]	O	Output data/status from RB to DMA controller
RB_DR_RDY	O	End of dword transfer 0 => DMUR should insert wait state. 1 => RB will finish transfer during this clock cycle.
RB_DR_STAT	O	Current dword is status type
RB_DR_REQ_N	O	Service request from RB to DMUR controller. Asserted when data available for transfer(burst threshold reached or status dword received from PMR).
Target FPI Slave Interface		
PB_TFPI_A[8:2]	I	Address bus.
PB_TFPI_D[31:0]	I	Input data bus.

PB_TFPI_WR_N
PB_TFPI_RD_N
	I
	I
		Read/Write controls. Following codes are defined:
		WR_N = 1; RD_N = 1 => NOP
		WR_N = 0; RD_N = 1 => data written to DMUR
		WR_N = 1; RD_N = 0 => data read from DMUR

PB_TFPI_RDY_EN	I	Enable start of bus cycle
PB_VC_TFPI_SEL_N	I	Select virtual channel registers block
PB_VG_TFPI_SEL_N	I	Select virtual global registers block
PB_RB_TFPI_SEL_N	I	Select RB specific registers block

Table A.3
RB Interfaces and Signal Descriptions for M256F

Symbol name	I/O	Function
RB_TFPI_RDY	O	Data cycle ending
RB_TFPI_D[31:0]	O	Output data bus.
Interrupt Controller Interface (IC)		
RB_IC_REQ_N	O	RB Interrupt Active
IC_RB_GNT_N	I	Interrupt acknowledged
IC_D[31:0]	I	Interrupt vector from macro in daisy chain
RB_IC_D[31:0]	O	Interrupt vector output next macro or interrupt controller
Data Buffer Data/Status RAM Interface (M256F)		
RB_DSR_WN	O	write strobe (active low)
RB_DSR_BSN	O	block select (active low)
RB_DSR_E(11:0)	O	address bus
RB_DSR_ED(31:0)	O	write data to ram
RBR_DSR_AZ(31:0)	I	read data from ram
Data Buffer Link RAM Interface (M256F)		
RB_DLR_WN	O	write strobe (active low)
RB_DLR_BSN	O	block select (active low)
RB_DLR_E(11:0)	O	address bus
RB_DLR_ED(16:0)	O	write data to ram
RBR_DLR_AZ(16:0)	I	read data from ram
Burst Parameter RAM Interface (M256F)		
RB_PTR0_WN	O	write strobe (active low)
RB_PTR0_BSN	O	block select (active low)
RB_PTR0_E(7:0)	O	address bus
RB_PTR0_ED(3:0)	O	write data to ram
RBR_PTR0_AZ(3:0)	I	read data from ram

Table A.3
RB Interfaces and Signal Descriptions for M256F

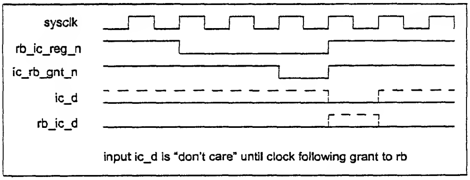
Symbol name	I/O	Function
Parameter Table RAM Interface		
RB_PTR1_WN	O	write strobe (active low)
RB_PTR1_BSN	O	block select (active low)
RB_PTR1_E(7:0)	O	address bus
RB_PTR1_ED(30:0)	O	write data
RBR_PTR1_AZ(30:0)	I	read data
Action Queue RAM Interface		
RB_AQR_WN	O	write strobe (active low)
RB_AQR_BSN	O	block select (active low)
RB_AQR_E(8:0)	O	address bus
RB_AQR_ED(16:0)	O	write data to ram
RBR_AQR_AZ(16:0)	I	read data from ram

A.4 Reset Adaptation

The M256F requires both a hardware and software controlled resets. A project specific macro reset block is provided which combines these resets into a single reset for the RB top level and is transparent to the RB core.

A.5 Interrupt Adaptation

The M256F maps the RB interrupts into a 32 bit vector that is combined with interrupt vectors from other macros in a daisy chained bus. A macro's interrupt vector output becomes the interrupt vector input of the next macro in the chain. That macro forms its output by the bit-by-bit 'or' of its interrupt vector with the input interrupt vector. A macro must hold its interrupt vector to all zeros until it is granted the bus to avoid disturbing other macros which may be using the bus.



input ic_d is "don't care" until clock following grant to rb

Figure A.4 RB Interrupt Bus Request/Grant Sequence

31															16
1	1	1	0	0	Queue ID(0-7)			0	0	0	0	0	RBI	AQI	0

15															0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure A.5 RB Interrupt Vector for generated for M256F application

A.6 FPI Adaptation to SMIF

The M256F uses 32 bit variant of the FPI bus. The bus is simplified and supports only overlapped read or write cycles.

The TFPI bus registers specify channel configuration and support TB testing. Addressing of channel specific data and all internal RAM is done using an indirect addressing scheme to minimize the required address space.

All addresses are dword aligned. Note that the address bus does support octet addressing (bits 0 and 1 are not implemented and are set to '0' internally).

The data output bus is a daisy chained bus (refer to previous section for basic concept). When the RB is not being addressed, it keeps its output data contribution to the daisy chain at all zeros so that it does not disturb other bus users.

A concept used in the M256F is the virtual register. The concept was developed to allow certain data which is distributed over more than 1 macro to be viewed by external software as single register. This hides internal macro function split from the user at the top level because such structure is a convenience for the chip developer and not of interest to the user. Internally, the FPI bus interface decodes certain address ranges corresponding to these virtual data blocks. Three select signals are generated for RB:

1. Virtual Channel register select (PB_VC_TFPI_SEL_N). Data in the block is channel specific and is distributed over all macros handling protocol channels. RB receives small slice of channel data (burst threshold parameter) which is captures on write of a specific row address of channel data. It returns this small slice on any read of the virtual register row address containing the parameter.
2. Virtual Global register select (PB_VG_TFPI_SEL_N). Data in this block is also distributed over multiple macros but is not channel specific. This data group includes system interrupt queue ID, masks for its interrupts, free pool monitoring modes, and test command for RB. The test command register uses another level of addressing via a macro id field. When the register is written with a valid macro id, the ownership of the test command and test data register passes to that id. The RB is assigned "0011" for M256F.
3. RB specific register select (PB_RB_TFPI_SEL_N). These are registers exclusive to the RB such as free pool counters.

The RB will not start FPI cycle until the input ready PB_TFPI_RDY is asserted. Once a bus cycle begins, it will extend its read ready cycle i.e. drive its output data bus until the input PB_TFPI_RDY goes high. This signal is the AND of all macro ready signals and stretches a virtual register read until the slowest macro data is available.

Table A.4 FPI Addresses for M256F Registers

Address Offset		Register Description
00 Hex PB_VC_TFPI_SEL_N must be asserted	W	Virtual Channel Register Command bits 0 - 7 : channel number 16 - 23 : command The following commands are supported: 1: initialize channel (accept burst parameter) 5: debug (writes channel number only) 6: no operation All other commands ignored.
20 Hex PB_VC_TFPI_SEL_N must be asserted	R/W	Virtual Channel Bus Burst Length bits 0-3 are burst parameter code: code dword burst length 0 1 1 4 2 8 3 12 4 16 5 24 6 32 7 40 8 48 9 64 10-15 are mapped to burst length 64

Address Offset		Register Description
40 Hex PB_VG_TFPI_SEL_N must be asserted	R/W	Configuration1: bit 2: Free Pool Monitor Mode 0 -> capture minimum free pool count 1 -> interrupt if free pool falls below threshold programmed in RB Threshold Register. Reset value is '0'. bit 5: RB Interrupt Mask 0 -> enable free pool monitor interrupt 1 -> disable free pool monitor interrupt Reset value is '0'.
44 Hex PB_VG_TFPI_SEL_N must be asserted	W	Configuration 2 bits 28-30: System Interrupt Queue ID RB sends free pool monitoring interrupts to this queue.
58 Hex Only valid if TFPI_SEL_VG_N is asserted	W	Test Command Register bits 0-11 : address field bits 16-23 : Command 0 - R/W Burst Threshold Parameter RAM 1 - R/W Parameter Table (except BTC) 2 - R/W Data Buffer RAM 3 - R/W Link RAM 4 - R/W Action Queue RAM 5 - Set test mode (manufacturing test) 6 - Reset test mode bit 24: auto increment mode following read or write of Test Data Register 0-> no post increment of address register 1-> post increment of address register bits 28-31: target address verification code. RB accepts command if code="0011"

Address Offset		Register Description
5C Hex PB_VG_TFPI_SEL_N must be asserted	R/W	Test Data Register. Read or write of this register reads or writes RAM specified in previous Test Command Register. If auto increment mode is active, address will be automatically advanced to next word address after access. Address wraps automatically at end of selected RAM. Width of data bus depends on command written to command register: 0 - 4 bits 1 - 31 bits 2 - 32 bits 3 - 12 bits 4 - 17 bits 5 - data register not used 6 - data register not used
B0 hex PB_RB_TFPI_SEL_N must be asserted	R	Free Pool Counters (read only) bits 0-11: current free data pool data count bits 12-15: reserved (set to zero) bits 16-23: free action queue free pool count bits 24-31: reserved (set to zero)
B4 hex PB_VC_TFPI_SEL_N must be asserted	R/W	Free Pool Interrupt Threshold Parameters bits 0-11: data buffer free pool threshold bits 12-15: reserved (set to zero) bits 16-23: action queue free pool threshold bits 24-31: reserved (set to zero) If free pool mode is "minimum pool capture", reading this register returns the minimum free pool counts observed since the last read. A read resets the counts to maximum. If free pool mode is "threshold interrupt", reading this register returns the current programmed threshold values.

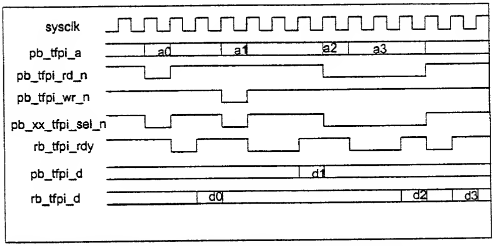


Figure A.6 FPI bus cycles : 1 read + 1 write + 2 overlapped reads

Appendix J

Title:
Protocol Machine Transmit Version 2.2

1 Introduction

1.1 Overview

The Protocol Machine Transmit (PT) provides channelized transmit protocol services for the following modes:

1. HDLC
2. bit-synchronous PPP
3. octet-synchronous PPP
4. Transparent

The PT architecture consists of a single logic core and a channel context memory for the channel's configuration and protocol state e.g. CRC sum, bit stuffing '1's count, interframe fill count, etc. The physcial context memory which is outside of the macro itself would be RAM when a large number of channels are supported. A register bank could be used when only a few channels are implemented. With this architecture, the core logic is essentially fixed except for channel address bus width. The application's clock speed and required aggregate throughput determine the maximum number of channels that can be supported.

Important features:

- HDLC, bit-synchronous PPP, octet-synchronous PPP, and Transparent modes
- Flexible scaling by dimensioning of context memory
- Per channel protocol configuration
- Configuration via Simplifed Microcontroller Interface (SMIF) registers
- Aggregate throughput of 52.8 MBit/second @ 33 MHz clock

1.2 System Integration

The PT has interfaces for :

- Configuration and Control : 32 bit Simplified Microcontroller Interface SMIF)
- Service Data Input : 32 bit Transmit Buffer (TB)
- Protocol Data Output : 8 bit data port for Timeslot Assigner Transmit (TT)
- Interrupts : end-of-frame or buffer under-run

The maximum aggregate throughput of 52.8 Mbit/second @ 33Mhz system clock is attained with:

- 1 octet of protocol data transferred to TT every 5 system clocks
- 1 dword from TB per 20 clocks on average.

Interrupts are used to report failures during protocol handling e.g., TB underrun or to report the end of a transmitted frame in order to coordinate release of transmit buffers.

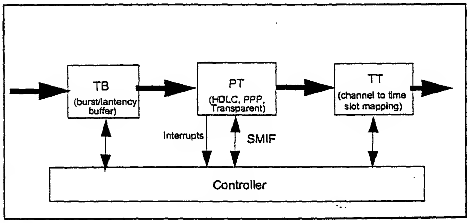


Figure 1 System Integration

1.3 Known Restrictions and Problems

2 Functional Description

2.1 Block Diagram incl. Clocking Regions

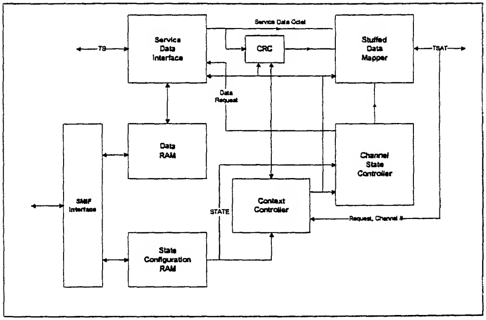


Figure 2
PT Bock Diagram

2.2 Normal Operation

2.2.1 Functions Common to all Protocols

To minimize the overhead of context switching, PT processes TT channel requests on octet sized data blocks. In order to provide flexibility to systems which use smaller blocks, the TT can also use a mask to enable individual bits in the octet. Unused bits of the octet are set to '1'. The mask essentially converts the TT interface from octet sized to N bits (0 < N <=8) per TT service. Although this adds flexibility, it should be used with caution because it reduces the aggregate. A mask is attractive for sub-channels defined in a timeslot because is greatly simplifies the TT design.

Note that TT could be replaced in an application with an interface to multiple, high or low speed serial data links by providing serial to parallel converters or converting the interface to bit serial operation via the mask. The channel number provided to PT would then correspond to link number. For historical reasons, Time Slot Assigner Transmit (TT) will continue to be used in this document although it is somewhat limiting in scope.

A pipelined design is used within PT to improve logic timing. Because PT channel processing is only performed in response to TT requests, the pipeline delays the start of a channel by several TT request cycels. After a channel is initialized via SMIF iterface, it remains in an idle state with all PT pipelines empty until the first TT request. PT responds as follows:

1. Load channel context into PT core logic.
2. Transfer protocol data in pipeline immediately to TT in bits specified by octet mask. If no protocol data available, transfer idle code or interframe fill or abort when open frame (under-run). Genererate Interrupt if under-run and the interrupt is enabled.
3. Return ready to TT.
4. If service data from TB is available in the PT's input buffer and the protocol pipeline has sufficient free space, generate new protocol data for the channel just serviced.
5. Save channel state back to context memory.
6. If PT's input buffer has free space for a double word, request data from TB.

PT provides the active status of a channel by asserting PT_TTCH_ON when it returns data to TT. Use of this status by TT is application specific e.g. TT might drive its outputs into tri-state for an uninitialized channel or it might simply ignore it and use data from PT transparently.

Protocol data for TT be be inverted on a per channel basis. This is useful for HDLC protocols over AMI links which cannot tolerate long strings of zeros. It converts the '0' stuffing of HDLC into '1' stuffing.

Status words are used to transfer data between the TB and PT when:

1. dword contains < 4 octets of service data or
2. end of frame for HDLC or PPP frame (FE=1) or
3. abort of open HDLC or PPP frame required (TAB=1).

A status word with BE=0 (no data) and neither flag set is invalid. The TBSTAT line is asserted high along with TBRDY=1 to indicate a status word.

Table 1 Status word

Bit	31	30	29..26	25	24	23..	..16	15..	..8	7..	..0
Function	FE	TAB	Reserved	BE		Byte2		Byte1		Byte 0	

If BE = 00 , the dword contains only status information. Otherwise BE is the number of valid service data octets starting at Byte 0.

TAB=1 causes an abort sequence to be sent if a frame is open. Otherwise it is ignored.

FE=1 indicates end of frame. FCS will be inserted followed by a closing flag.

2.2.2 HDLC mode

PT performs only the low level functions of frame generation i.e. flag insertion, CRC generation, and bit stuffing.

Flag	Address	Control	Information	CRC	Flag
0111 1110	8 bits	8 bits	<=0 Bits	16/32 bits	0111 1110

Figure 3 HDLC Frame Format

The frame start and end is marked with the flag character (7Eh). Interframe time fill character of either 7Eh or FFh is sent when a frame is not open. The mimimun number of interframe characters to be transmitted is controlled by the parameter FNUM (0<=FNUM<=255). The use of FNUM is:

FNUM = 0 : shared closing and opening flag l

FNUM /= 0 : if interframe fill character is 7Eh (flag),

# of interframe characters = (1+ FNUM) flags

else (interframe fill character is FFh)

# of interframe characters = 2flags + FNUM 0FF hex characters

A shared 0 bit between two flags is not supported in the PT.

When PT reads service data from TB, a new frame is automatically started after the programmed number of interframe fill characters have been sent. **Note that FNUM is not used at the initial start of a channel i.e. the first frame is sent without delay.** The Frame Check Sequence (CRC16 or CRC32) is initialized at the beginning of the frame and then updated for all service data in the frame. Zero bit insertion (stuffing) is performed by inserting a '0' bit after all sequences of five contiguous '1' bits in the service data and the final FCS.

If an underrun occurs in TB, it asserts the TB_EMPTY line along with TBRDY at the end of a PT read cycle. When this happens in the middle of a frame, an abort sequence "01111111" is automatically generated. An underrun interrupt is generated when it is enabled.

The abort sequence is also generated but without an interrupt when the Transmit Abort falg TAB is set in the status word. This flag would be used when removing a channel from service or interrupting an open frame. An TAB is ignored if a frame is not open.

When the frame end is reached (FE=1 in the status word), the FCS is transferred to TT after bit stuffing, the closing flag inserted, and the interframe fill mode entered. A new frame can be started as soon as the interframe fill count is satisfied.

2.2.3 Bit synchronous PPP

The mode is identical to the HDLC mode except for the abort sequence when the interframe fill pattern is 0FF hex. When 7Eh is programmed as interframe time fill character, the abort sequence consists of "01111111" as in HDLC. When 0FFh is programmed, the abort sequence consists of 15 '1's.

2.2.4 **Octet Synchronous PPP** (OSPPP)

OSPPP uses a frame structure similar to the HDLC mode:

- frame start and end synchronization is performed using the flag character (7Eh).
- 16 or 32 bit CRC is computed over all service data read from TB and appended to the end of the frame.
- interframe fill character is used but will always be 7E hex (flag). the use of FNUM is identical to HDLC case for 7E fill pattern.

OSPPP differs by using octet stuffing of service data and FCS instead of '0' bit stuffing. Octet stuffing replaces certain specified octet values with a 2 octet sequence consisting of $7D_{Hex}$ (Control Escape) followed by the octet EXOR'ed with 20_{Hex} (e.g. 13 Hex is mapped to 7Dh, 33h). OSPPP allows suppression of special control characters used by transmission equipment such as modems. These characters can be transparently inserted because they can be recognized as spurious and removed at the end point OSPPP handler since valid data will be substituted. **Note that this mode can only be used on links which are character oriented.** Otherwise, adjacent characters could inadvertently form a flag in the bit sequence e.g. 57 + E9 hex This precludes use of OSPPP on sub-channels (mask /= 0FF hex) because the characters can't be synchronized except by the time slot boundary.

Characters which are stuffed are :

- Control Escape $7D_{Hex}$
- $7E_{Hex}$ (flag character) in service data or FCS (not opening or closing flags)

- DEL control character (optional)
- characters specified in 32 bit Asynchronous Control Character Map (ACCM)(optional)
- characters specified in 4 bit Extended ACCM (EACCM)(optional)

The substitution of Control Escape and flag characters is mandatory and so doesn't require enabling flags. The ACCM, EACCM, and DEL enables are per channel. The ACCM enables stuffing for all characters in the range $00\text{-}1F_{Hex}$. EACCM enables stuffing of 4 ACCM extension characters which are specified characters in a user programmable 32 bit (4 character) register common to all channels. The opening and closing flags are not stuffed.

The OSPPP abort sequence consists of the Control Escape character 7D hex followed by a flag character $7E_{Hex}$ (not stuffed). Aborts are generated identically to HDLC cases.

Between two frames, the interframe time fill character is always 7Eh. The count of interframe fill characters is set by FNUM as described in the HDLC description.

2.2.5 Transparent Mode (TM)

In the transparent mode, PT performs data transmission without any framing, i.e. without

- Flag insertion
- CRC generation
- Bit or octet stuffing

The data read from TB is simply buffered in PT and copied to TT after the channel is synchronized. Synchronization is provided to support super-channels (fractional T1 or T1) for TM. The start of the transmission of data is delayed by PT until the time marker TT_SYNC is asserted by TT at the start of its read cycle. Normally this would occur at the first time slot of a superchannel. After initialization in TM, PT remains in a waiting state where it transfers all ones data to TT. When TT_SYNC line is asserted and the PT pipeline has protocol data, the channel is synchronized and begins transferring data.

After the first activation of TT_SYNC, PT ignores TT_SYNC and transfers data to TT until an underrun occurs. TM does not have an interframe time i.e. once the channel is synchronized, it must always receive data from TB without interruption. A Frame End flag is not valid. Underrun causes PT to go back to the waiting state until TT_SYNC is again asserted. An underrun generates an interrupt when enabled.

A programmable TM Flag (TFLAG) character can be sent whenever service data is not being transferred (TM interframe fill character).

An 'Transparent Mode Pack' option is provided to support subchanneling. When subchanneling is used (logical channels of less than 64 kbit/s), service data is delivered only in bit positions marked by a '1' in the mask from TT. Unused bits are fixed at '1'. The user would normally send packed data to PT maps it into the enabled bit positions and inserts '1' bits in bit positions not enabled. If the mode is not packed, the user service data is transferred to TT regardless of the mask. The assumption is that the user has already unpacked the data and inserted the '1' bits at unused bit positions. This mode is not recommended and provided only for compatibility reasons.

2.3 Configuration and State RAMs

The following tables describes the configuration and state RAMS used by the PT

Table 2 Channel Configuration RAM layout (CN-1 words x 59 bits)

Bit / Field Name	# Bits	Description
ACCM	32	- 32 bit Async Character Control Map for OSPPP Mode - 8 bit TFLAG for Transparent Mode (lower 8 bits)
CH_EN	1	Enable channel
MODE	2	Protocol Mode (HDLC, Bit PPP, Octet PPP, Transparent)
CRC_DIS	1	CRC disabled
CRC32	1	CRC mode (0 -> CRC16, 1->CRC32)
TMP	1	Transparent Mode Pack 0 -> data octet transferred independent of enable mask 1 -> data octet unpacked as per enable mask (normal)
INVERT	1	Invert protocol data to TT
FA	1	Transparent Mode Flag Adjust 0 -> send FF in exceptions state (idle/waiting for sync) 1 -> send user specified TFLAG in exception state
IFTC	1	Interframe Time Character (HDLC or Bit Oriented PPP) 0 -> 7E hex 1 -> FF hex
ACCM_EXT	4	Extended ACCM map for user characters (OSPPP)
MAP_DEL	1	Enable DEL character octet stuff (OSPPP)
DIS_FE_INT	1	Disable Frame End interrupt
DIS_UR_INT	1	Disable Underrun Interrupt
INT_QUEUE_ID	3	Channel Interrupt Queue Identifier
FNUM	8	Interframe fill count (0-255)

Channel State RAM (CN-1 words x 105 bits)

Bit / Field Name	# Bits	Description
ROCTS	32	Remaining Octets in service wordto be translated
CRC	32	Frame Check Sequence being generated for open frame

Channel State RAM (CN-1 words x 105 bits)

Bit / Field Name	# Bits	Description
RBITS	17	Remainding protocol bits after transfer to TT: At least 8 bits must be available to immediately service TT on demand. When 7 remain, protocol data is generated at the end of TT service. For HDLC, 8 bit octet can be stuffed twice => 7+8+2 = 17 bits required.
RBITS_CNT	5	Number of valid RBITS
ONES_COUNT	3	Number of contiguous '1' s (0-4) for HDLC stuffing, flag, or abort detection.
STATE	3	Channel state
ROCNT	3	Number of valid ROCTS (0-4)
STAT	1	STAT flag for ROCTS context
NO_DS	1	flag indicating no data or status available in context
DLY_CH_ON	1	Pipelined (delayed) flag to turn on/off the channel
CCNT	8	Control Counter (dual use) a. control insertion of CRC octets at frame end b. count interframe fill characters
PPP_CTRL	1	OSPPP octet stuffing flag

2.4 Reset Behavior

All internal registers and functions are asynchronously reset to defined states. The RESET signal must be removed synchronously to ensure reliable operation.

After reset the reception of data is turned off for all the channels and all registers are accessible via SMIF registers for initialization.

3 Macro Interfaces and Signal Description

All signals are active high until otherwise specified. Active low signals are designated by "_N" appended to their names. To make the design as re-usable as possible, bus signals with application dependent width are specified with one of the following parameters:

Parameter name	Bus Type
CN	Channel Number Bus
DB	Data/Status Bus

3.1 Signal Description

Table 3
Interfaces and Signal Description

Symbol name	I/O	Function
Clock and Reset		
SYSCLK	I	Clock
RESET_N	I	General reset of PT. All registers and RAM reset
STOP	I	Blocks PT from processing new TT requests
PT_IIP	O	PT initialization active. Writing all RAMs to defined state
Timeslot Assigner Transmit (TT) Interface		
TTRD_N	I	TT data request.
TTCH[CN-1:0]	I	channel number
TTMASK[7:0]	I	enable mask. mask bit N enables bit N in PT_TTD '0' => PT_TTD[N] not used. PT_TTD[N] = '1' '1' =>PT_TTD[N] is protocol data
TTSYNC	I	channel synchronization line (Transparent Mode only)
PT_TTRDY	O	PT will be able to finish transaction in current clock cycle

Table 3
Interfaces and Signal Description (cont'd)

Symbol name	I/O	Function
PT_TTCH_ON	O	Channel Operating Status 0 => channel is off. PT_TTD is not valid. 1 => channel is on. PT_TTD is valid.
PT_TTD[7:0]	O	Protocol data octet

Transmit Buffer (TB) interface

PT_TBRD_N	O	Request TB read
PT_TBA[CN-1:0]	O	Channel Number for TB read request
TBRDY	I	TB completing read cycle this clock
TBD[DB-1:0]	I	Read data. Valid when TBRDY='1'
TBSTAT	I	TBD data type '0' => TBD is DB/8 protocol data octets '1' => TBD is status word . TBSTAT is valid when TBRDY='1'
TBEMPTY	I	TB empty flag '0' => TBD is valid '1' -> No data or status available for channel PT_TBA. TBEMPTY is valid when TBRDY='1'

Interrupt Interface

PT_URUN_INTRPT	O	Interrupt :TB Underrun during open frame (error)
PT_FE_INTRPT	O	Interrupt : Frame End (not error)
PT_INTRPT_CN[CN-1:0]	O	Channel number
PT_INTRPT_QID[3]	O	Interrupt Queue ID Per channel user specified parameter
INT_ACK	I	Interrupt service acknowlegement

SMIF Interface

Table 3
Interfaces and Signal Description (cont'd)

Symbol name	**I/O**	**Function**
BPI_RD_SFR_N[7:1]	I	Read special function registers (1 select per register) **Bit** **Function** 1 : channel mode 2 : channel Asynchronous Control Character Map 3 : Extended Asynchronous Control Characters 4 : channel interrupt priority (queue ID) 5 : channel interrupt masks 6 : test access command register 7 : test access data register Note that bit select 0 is not implemented because command register is write only.
BPI_WR_SFR_N[7:0]	I	Write special function registers (1 select per register) **Bit** **Function** 0 : channel command register 1 : channel mode 2 : channel Asynchronous Control Character Map 3 : Extended Asynchronous Control Characters 4 : channel interrupt priority (queue ID) 5 : channel interrupt masks 6 : test access command register 7 : test access data register
BPI_REQ_N	I	BPI Request '0' => BPI_RD_SFR_N or BPI_WR_SFR_N valid '1' => BPI_RD_SFR_N or BPI_WR_SFR_N not valid
BPI_DI[DB-1:0]	I	SMIF write data input bus
BPI_RDY_N	O	SMIF read/write cycle ending this clock
BPI_D_O[DB-1:0]	O	SMIF read data output bus

3.2 Data Flow and Functional Timing

3.2.1 Timeslot Assigner (TT) Interface

The transfer is initiated by TT by asserting TTRD_N='0' along with channel number and mask lines. PT returns protocol data when the PT_TTRDY line is active.

TT must also activate the TTSYNC line during the first timeslot of a superchannel.

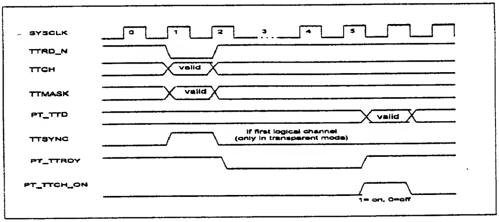


Figure 4
Data transfer from PT to TT

3.2.2 TB Interface

PT initiates an address cycle by asserting channel address PT_TBA and read request PT_TBRD_N. TB must capture the channel number from the address bus during this cycle. During the data phase, TB asserts TBRDY, TBSTAT, TBEMPTY, and TBD at the beginning of the clock cycle that the data is available. TB may insert wait states but the response time must be appropriate to the application. **A maximum of 4 wait states is allowed in any application. If TB uses 4 wait states, it must be capable of accepting a new read cycle simultaneously with the assertion of its TBRDY.**

TBEMPTY is used to stop a PT transfer when the TB has no service data for the addressed channel. TB terminates the cycle normally with TBRDY but asserts TBEMPTY to show that the data is not valid.

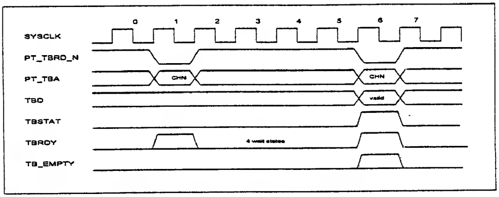


Figure 5 PT read cycles: 1 success and 1 interrupted transfer

4 Special Function Registers Description

4.1 Channel Command Register

31								16
CMD_XMIT(7:0)	0	0	0	0	0	0	0	0

15							8	7 0
0	0	0	0	0	0	0	0	CHANNAL

CHANNAL specifies the channel number to be programmed or to be read.

The Channel Command Register can be used in a "broadcast" environment where a command may be written to more than 1 macro and each macro responds appropriately and in parallel. The actual binary value for each command is set by parameters in the VHDL package at synthesis time.

The commands supported by PT are:

1. Initialize

   Copies channel parameters from SMIF registers into context memory.
   Activates channel.

2. Channel Off

   Deactivates channel after channel's protocol data is flushed from PT context memory.

3. Initialize/Update

   For PT, identical to Initialize command. Update PT parameters after initialization. Provided for systems where Initialize command shared by other macros that cannot update their parameters.

4. Transmit Debug

   Loads channel number for subsequent read of channel parameters.

5. Send Idle

   Deactivates channel if it is on. Send idle code specified in FNUM parameter of configuration data.

For any other command, PT returns ready but ignores the command.

4.1.1 Channel Configuration

Access	: read/write
Reset Value	: 00000000_H

31	
FNUM(7:0)	TFLAG(7:0)

15											0
IFTF	0	FA	INV	TMP	0	CRC 32	CRC DIS	EACCM (3:0)	DEL	0	PMD(1:0)

PMD(1:0): Channel Mode
00 : HDLC
01 : BSPPP (bit synchronized)
10 : OSPPP (octet synchronized)
11 : Transparent

DEL: Del Character map enable flag (OSPPP mode only)

EACCM : Extended ACCM for 4 user specified characters (OSPPP mode only)
0 => disables mapping of respective character in EACC register
1 => enables mapping of respective character in EACC register

CRCDIS: Disable Frame Check Sequence (CRC) for HDLC or PPP
0 => CRC is inserted at end of frame
1 => CRC is not inserted at end of frame

CRC32: Enable CRC32 FCS
0 => 16 bit CRC is enabled
1 => 32 CRC is enabled

TMP: Transparent Mode Pack mode (transparent mode only)
0 => TT mask OR'ed with service data octet (no unpacking by PT)
1 => service data unpacked by PT and distributed according to TT mask.

INV: Invert channel protocol data (idle pattern when channel is off not affected)
0 => no inversion

1=> invert channel' s transmit protocol data

FA: Transparent Mode fill pattern (exception conditions)

0 => insert 0FF hex when channel is idle or not synchronized
1 => insert TFLAG when channel is idle or not synchronized

IFTF: Interframe Time Fill Character (HDLC or BSPPP only)

0 => $7E_H$
1 => FF_H

TFLAG : Transparent Mode Flag (Exception Condition Fill Pattern only if FA=1)

8-bit "fill flag" for transparent mode. It is inserted into transmit data when FA=1 and channel is idle or not synchronized in transparent mode.

FNUM : Minimum number (0-255) of interframe fill characters (HDLC or PPP only).

4.1.2 Channel Asynchronous Control Character Map (ACCM)

Access : read/write

Reset Value : 00000000_H

31															16
1F	1E	1D	1C	1B	1A	19	18	17	16	15	14	13	12	11	10

15															0
0F	0E	0D	0C	0B	0A	09	08	07	06	05	04	03	02	01	00

Used in octet synchronous PPP mode only. When bit N is set in the map, character with value N (00 to 1F) replaced by Control ESC sequence in the outgoing data stream.

4.1.3 Extended Asynchronous Control Characters

Access : read/write

Reset Value : 00000000$_H$

31	16
CHAR3(7.0)	CHAR2(7:0)

15	0
CHAR1(7:0)	CHAR0(7:0)

This register is common to all octet synchronous PPP mode channels. When enable flag is set in the channel's XACCM (4 bits in Channel Configuration), the corresponding character written in this register will be replaced with a Control Escape sequence. This extends the basic 32 character ACCM with 4 user specified characters.

4.1.4 Channel Interrupt Priority

Access : read/write

Reset Value : 00000000$_H$

31	
TQUEUE(2:0)	not used.

15	0
not used	

TQUEUE : channel interrupt priority

PT does not use this parameter directly but stores it and provides it when it generates an interrupt . The interrupt controller can use the parameter to prioritize the interrupt.

4.1.5 Channel Interrupt Masks

Access	: read/write
Reset Value	: 00000000_H

31								23	22						16
0	0	0	0	0	0	0	0	UE	FE	0	0	0	0	0	0

15															0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Interrupt masks disable their respective interrupt when set high.

FE : disable Frame End interrupt
UE : disable channel data Under-run Error

4.1.6 Test Command Register (TAC)

Access	: write
Reset Value	: 00000000$_H$

31					
MID	0	0	0	AI	CMD

15	0
ADDRESS	

MID:	Macro ID Code
AI:	Auto Increment Function
Address:	RAM read/write address bits
CMD:	Command

The test access provides read/write access of PT state memory. Because the state bus is 105 bits (refer to previous state description), 4 commands are implemented to read it in SMIF sized slices. PT accepts a test command only when the MID code matches PT ID code (assigned at synthesis). Defined command opcodes are:

CMD	read/write of state ram bits
0	0-31
1	32-63
2	64-95
3	96-104

If AI (autoincrement)=1, a post increment of ram address is performed to simplify block read or write operation.

To perform test access, TAC is written and then TAD is written or read.

4.1.7 Test Data Register (TD)

Access : read/write
Reset Value : 00000000_H

31
TEST DATA

15	0
TEST DATA	

Refer to Test Command Register for a description of data.

A-1 Introduction to M256F Application

A-1.1 Overview

The important M256F characteristics are:

- 256 channels
- aggregate throughput of approximately 50 Mb/s @33Mhz clock
- HDLC, bit-synchronous PPP, octet-synchronous PPP, Transparent modes

These requirements can be met without changes to the PT core macro except for changes in constants e.g. M256F definitions for channel address bus width, channel command opcodes, and macro ID. An application shell adapts the SMIF interface to the Flexible Peripheral Interface (FPI) bus and the PT's simple interrupts to the M256F's interrupt vector bus. An interrupt buffer is implemented in the shell due to the latency of the M256F interrupt controller.

A-1.2 System Integration

The main PT interfaces are:

- FPI Slave Bus (PT SMIF adaptation in shell)
- Timeslot Assigner Transmit (TSAT) (no adaptation required)
- Transmit Buffer (TB) (no adaptation required)
- Interrupt Controller (vector and service latency adaptation in shell)

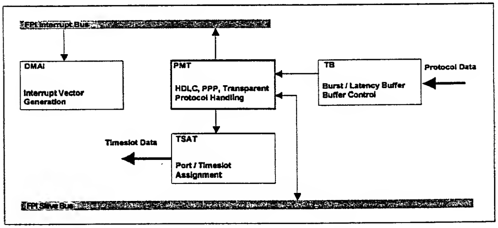

FPI Interrupt Bus
DMAI
Interrupt Vector Generation
PMT
HDLC, PPP, Transparent Protocol Handling
TB
Burst / Latency Buffer Buffer Control
Protocol Data
Timeslot Data
TSAT
Port / Timeslot Assignment
FPI Slave Bus

Figure A-1 System Integration in M256F

A-2 M256F PT Application Shell

The M256F application required adaptation of the PT in the following areas:

1. Configuration and control. Adapt SMIF to FPI bus which uses daisy chain concept to reduce bus area and avoid 3-state buses.
2. Interrupts. Adapt simple interrupt to a system conforming interrupt structure and daisy chain bus.

The reset also required adaption of the PT's single interrupt to separate hardware and software reset inputs. This was implemented transparently to the PT by using a project standard reset core.

The M256F uses a daisy chained bus structure for interrupts and configuration/control. In this concept, a macro's output bus becomes the input bus for the next macro in the chain. A macro forms its daisy chain output by 'or'ing, bit-by-bit, its output data with the input bus. To avoid disturbing other bus users, a macro holds its output data to all zeros until it is granted the bus.

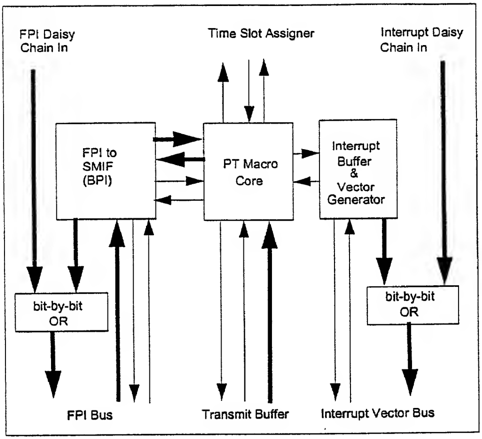


Figure A-2 Structure of M256F PT Application Shell

A-2.1 Transmit Buffer Adaptation

No shell adaptation other than renaming of signal names to satisfy M256F naming convention is required.

A-2.2 Time Slot Assigner Adaptation

No shell adaptation other than renaming of signal names to satisfy M256F naming convention is required.

A-2.3 Control and Configuration (SMIF) Adaptation

The M256F uses a simplified 32 bit variant of the FPI bus which supports overlapped read and write cycles only. FPI registers are dword aligned. The address bus does support octet addressing (bits 0 and 1 are not implemented).The data bus is daisy chained (see previous section).

The M256F uses a concept called the virtual register. The concept allows a data register which is distributed over more than one macro to be viewed by external software as a single register. This hides the internal macro function split which is not relevant to the user. Internally, the FPI bus interface decodes certain address ranges corresponding to these virtual data blocks. Three select signals are generated for PT:

1. Virtual Channel register select (PB_VC_TFPI_SEL_N). Data in this block is channel specific and is distributed over all macros handling protocol channels. PT captures channel configuration data located at several offset addresses in the channel register block. After a channel is initialized, PT returns this data when the channel's virtual channel register containing it is read.
2. Virtual Global register select (PB_VG_TFPI_SEL_N). Data in this block is also distributed over multiple macros but is not channel specific. This data group includes system interrupt queue ID, interrupt masks and test commands for PT. The test command register uses another level of addressing via a macro id field. When the register is written with a valid macro id, the ownership of the test command and test data register passes to that id. The PT is assigned "0010" for M256F.
3. PT specific register select (PB_PT_TFPI_SEL_N) for registers exclusive to the PT. Only the Extended Asynchronous Control Characters register is in this catagory.

The PT will not start an FPI cycle until the input ready signal PB_TFPI_RDY is asserted. Once a bus cycle begins, PT extends its read ready cycle i.e. drives its output data bus until the input PB_TFPI_RDY goes high. This signal is the "AND" combination of all macro ready signals and thus stretches a virtual register read until data from the slowest macro is available.

A-2.4 Interrupt Adaptation

The M256F maps interrupts into a 32 bit vector that is combined with interrupt vectors from other macros in a daisy chained bus. Each macro's interrupt vector output becomes the interrupt vector input of the next macro in the chain. The macro forms its output by the bit-by-bit "OR" of its own interrupt vector with the input interrupt vector. The macro must hold its interrupt vector to all zeros until it is granted the bus to avoid disturbing other macros which may be using the bus.

PT generates data UnderRun (UR) and Frame End (FE) interrupts for the channel. The interrupt vector includes a constant identifying the interrupt vector type, the 2 interrupt flags, the interrupt priority (queue ID), and the channel number.

The PT shell provides a 32 entry buffer for interrupt requests which is necessary to handle the interrupt controller's latency. In the worst case, PT can generate an interrupt every 5 clock cycles. Although the interrupt controller can handle this on average, the collisions with other bus users will delay service. The buffer stores the interrupt flags and channel number and generates the interrupt vector on the fly.

Table A-1 PT Interrupt Vector

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
1	0	1	1	0	Queue ID (0-7)			0	0	0	0	0	0	0	0

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
UR	FE	0	0	0	0	0	0	Channel Number(0-255)							

A-3 Interface Signals Description

All signals are active high until otherwise specified. Active low signals are designated by "_N" appended to their names.

Table A-2 Interfaces and Signal Description

Symbol name	I/O	Function
Clock and Reset		
SYSCLK	I	Internal clock (33 MHz) derived from FPI master bus
HW_RESET_N	I	Asynchronous reset of PT
SW_RESET_N	I	Software controlled reset of PT
Timeslot Assigner (TT) Interface		
TT_PT_RD_N	I	TT beginning read of channel protocol data (1 clock wide pulse). All TT inputs to PT must be valid during the time this signal is asserted. They will be sampled at the rising edge.
TT_PT_CH[7:0]	I	Logical channel number.
TT_PT_MASK[7:0]	I	Enable mask for PT_TT_D . Function of bit N (0-7) is: 0 => bit N is not used (set to '1' by PT). 1 => PT inserts protocol data in bit N.
TT_PT_SYNC	I	Synchronization for Transparent Mode SuperChannel
PT_TT_RDY	O	PT will complete transfer in current clock cycle
PT_TT_D[7:0]	O	Protocol Data as per TT_PT_MASK
PT_TT_CH_ON	O	Channel on/off status. 0=> channel is off. 1=> channel is on.
Transmit Buffer (TB) Interface		
PT_TB_RD_N	O	PT read request for channel PT_TB_A service data.
PT_TB_A[7:0]	O	Channel number (0-255) for read request
TB_PT_D[31:0]	I	Channel data
TB_PT_RDY	I	TB will complete data transfer this clock

Table A-2 Interfaces and Signal Description (cont'd)

Symbol name	I/O	Function
TB_PT_STAT	I	Data type being read. Valid when TB_PT_RDY='1' and TB_PT_EMPTY='0'. 0 => 4 octets of service data. 1 => 0-3 octets of service data+ status octet
TB_PT_EMPTY	I	Transmit buffer empty flag. Valid when TB_PT_RDY='1' 0 => TB_PT_D is valid data/status. 1 => No data available for channel. TB_PT_D not valid.

Interrupt Controller Interface

PT_IC_REQ_N	O	PT request for interrupt bus PT_IC_D 0 => PT requesting bus (active until IC_PT_GNT_N='0') 1 => PT not requesting bus
PT_IC_D[31:0]	O	Daisy chained interrupt bus output
IC_PT_GNT_N	I	Interrupt bus grant. 0 => PT granted PT_IC_D bus during next clock. 1 => PT not granted PT_IC_D. During next clock, PT_IC_D=IC_D.
IC_D[31:0]	I	Daisy chained interrupt bus input

Target FPI Slave Bus

PB_TFPI_A[8:2]	I	Address bus.
PB_TFPI_D[31:0]	I	Daisy chained bus input data
PT_TFPI_D[31:0]	O	Daisy chained bus output data

TFPI_WR_N
TFPI_RD_N
	I
	I
		Read/Write controls. Following codes are defined:
		WR_N = 1; RD_N = 1 => NOP
		WR_N = 0; RD_N = 1 => data written to DMUR
		WR_N = 1; RD_N = 0 => data read from DMUR

TFPI_VC_SEL_N	I	Virtual Channel register select.
TFPI_VG_SEL_N	I	Virtual Global register select.
TFPI_PT_SEL_N	I	PT macro specific register select.

Table A-2 Interfaces and Signal Description (cont'd)

Symbol name	I/O	Function
PT_TFPI_RDY	O	End of transfer indicator: 0 => PT not ready. Master must insert wait states 1 => end of transfer during this clock.
PB_TFPI_RDY	I	Ready enable. PT will not recognize *_SEL_N input signal or end its own ready cycle unless this signal is asserted high. 0 => delay start or completion of PT bus cycle. 1 => enable start or completion of PT FPI bus cycle.

A-4 Data Flow and Functional Timing

A-4.1 FPI Slave Bus

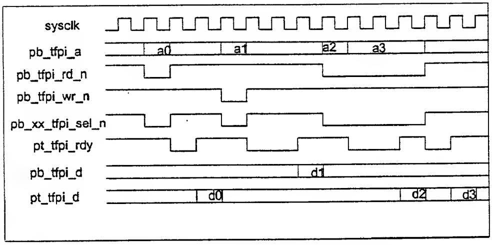


Figure A-3 FPI bus cycles : 1 read + 1 write + 2 overlapped reads

A-4.2 Interrupt Bus Interface

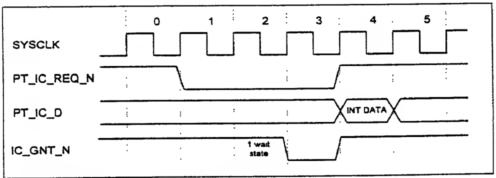


Figure A-4 Interrupt bus request/grant sequence

A-5 Register Description

A-5.1 Command Register

Access	: read/write
Address	: 00_H
Reset Value	: 00000000_H

PB_VC_TFPI_SEL_N must be '0' to access this register.

31	16
CMD_XMIT(7:0)	don't care

15 8	7 0
don't care	CHAN(7:0)

Commands affect the Virtual Channel Registers (VCRs). When programming a channel, the command should be written only after all required parameters have been written to the VCRs. In the case of a debug command which reads a channel's programmed configuration, the command register is written first and then the VCR's address is read. For a debug read, PT will "or" data into the daisy chain bus only for those bits in the register it "owns" i.e. captured during programming phase. Otherwise it passes daisy chain input bus to the output unchanged

Transmit commands recognized by PT:

00000000 : No operation
00000001 : Init Channel. copies previouslly written VC registers to channel state ram.
00000010 : Channel off. PT turns off channel after clearing protocol pipeline.
00010000 : Debug. channel address accepted for virtual channel register read access.
00100000 : Transmit Idle. Update configuration data TFLAG. Send TFLAG (idle).
01000000 : Update. Alias for Init Channel command. Allows interworking with other macros that cannot execute Init Channel a 2nd time without disturbing the channel. PT requires this function to update FNUM.

The Transmit Idle is used to send PCM idle code. This command turns off the channel's TB interface and sends the Transparent mode idle flag TFLAG to the Time Slot Assigner (TT). Idle code will not be correctly sent unless TT mask is all "ones".

A-5.2 VC Channel Configuration Register

Access	: read/write
Address	:14_H
Reset Value	: 00000000_H

.

PB_VC_TFPI_SEL_N must be '0' to access this register.

31

FNUM(7.0)	TFLAG(7:0)

15 0

IFTF	0	FA	INV	TMP	0	CRC 32	CRC DIS	EACCM (3:0)	DEL	0	PMD(1:0)

Refer to layout of Channel Configuration in description of RB macro SMIF registers.

A-5.3 VC Asynchronous Control Character Map (ACCM) Register

Access	: read/write
Address	: 18_H
Reset Value	: 00000000_H

.

PB_VC_TFPI_SEL_N must be '0' to access this register.

31 16

1F	1E	1O	1C	1B	1A	19	18	17	16	15	14	13	12	11	10

15 0

0F	0E	0D	0C	0B	0A	09	08	07	06	05	04	03	02	01	00

This map is used by channel for octet synchronous PPP mode only.When a bit is set, the corresponding character will be replaced by the Control ESC character in the outgoing data stream

A-5.4 VC Interrupt Queue ID (PT Interrupt Priority) Register

Access	: read/write
Address	: 20_H
Reset Value	: 00000000_H

PB_VC_TFPI_SEL_N must be '0' to access this register.

31	
TQUEUE(2:0)	don't care

15	0
don't care	

TQUEUE: PT interrupt vectors will contain this parameter as their queue ID.

A-5.5 VC Channel Interrupt Masks Register

Access	: read/write
Address	: $2C_H$
Reset Value	: 00000000_H

PB_VC_TFPI_SEL_N must be '0' to access this register

31	23	22	
Don't care	UE	FE	Don't care

15	0
Don't care	

PT nterrupt Masks:
0 => interupt is enabled.
1 => interrupt is disabled.

UE : UnderRun Interrupt Disable.
FE : Frame End Interrupt Disable

A-5.6 Extended Asynchronous Control Characters Register

Access : read/write
Address : 38_H
Reset Value : 00000000_H

PB_PT_TFPI_SEL_N must be '0' to access this register.

31	16
CHAR3(7:0)	CHAR2(7:0)

15	0
CHAR1(7:0)	CHAR0(7:0)

This register is only used by a channel in octet synchronous PPP mode. A character written to this register will be replaced with a Control Escape sequence when the corresponding enable flag is set in EACCM (3:0) field of channel configuration register.

A-5.7 Test Command and Address Register (TAC)

Access : read/write
Offset Address : 58_H
Reset Value : 00000000_H

PB_VG_TFPI_SEL_N must be '0' to access this register.

31

MID	0	0	0	AI	command

15 0

0	0	0	0	ADDRESS

MID : Macro ID Code. Must be 0010 to be accepted by PT.

AI : Auto Increment Function (enables post incress of ADDRESS following read or write access of Test Data Register.

Address : PT state ram address

Command : Specifies segment of PT state ram to read or write

Test registers are virtual global registers The function of test commands is explained in the RB macro description of the SMIF test register.

A-5.8 Test Data Register (TD)

Access : read/write
Address : $5C_H$
Reset Value : 00000000_H

PB_VG_TFPI_SEL_N must be '0' to access this register.

31

TEST DATA

15 0

TEST DATA

TD returns data read via TAC register.

Appendix K

Title:
Macro Specification PMR Version 2.1

1 Introduction

1.1 Overview

The Protocol Machine Receive (PMR) provides protocol handling for a flexible number of logical channels. The PMR implements 4 modes, which can be programmed independently for each channel: HDLC, bit-synchronous PPP, octet-synchronous PPP and Transparent.

The configuration of each logical·channel is programmed via the FPI slave bus and stored in the CSR - Context and Status RAM (or registers depending on the number of supported channels). The current state for the protocol processing (CRC check, channel state,...) is also stored in the CSR.

For the M256F, the performance requirements are 256 channels and a maximum serial rate of 43 Mbit/s. With an internal clock of 33 Mhz, 6 clock cycles are available in the average for loading of context, protocol handling of 8 bits and saving of context.

1.2 Features

- Protocol handling for a flexible number of logical channels
- Protocols supported: HDLC, bit-synchronous PPP, octet-synchronous PPP and Transparent mode
- Independent channel protocol configuration

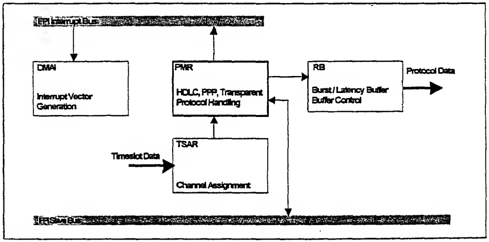
1.3 System Integration

The PMR has four interfaces:

- FPI Slave Interface
- Timeslot Assigner (TSAR) Interface
- RB Interface
- Interrupt Bus

The PMR receives Timeslot data over the TSAR Interface and transmits the protocol data over the RB Interface. If during the Protocol handling a failure occurs (e.g., RB overflow or short frames) an interrupt vector is generated and written on the interrupt bus.

The PMR is configured (HDLC, PPP or Transparent mode) via the FPI slave bus.



FPI Interrupt Bus
DMAI
Interrupt Vector Generation
PMR
HDLC, PPP, Transparent Protocol Handling
RB
Burst / Latency Buffer Buffer Control
Protocol Data
TSAR
Channel Assignment
Timeslot Data
FPI Slave Bus

Figure 1
System Integration

1.4 Known Restrictions and Problems

2 Functional Description

2.1 Block Diagram incl. Clocking Regions

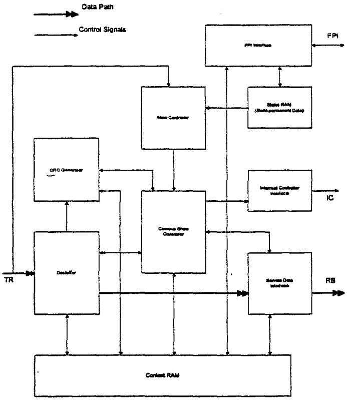


Figure 2
PMR Bock Diagram

2.2 Normal Operation Description

The TSAR will start a transfer to PMR by activating its TRWR_N line and providing the channel number (like the address bus on the FPI bus). In the second cycle TSAR will provide 8 bits data and the mask field (as bit enable lines). The PMR will load the context of the channel in the CSR, handle the 8 bits data and store the actual context again in CSR. Once 32 bits have been processed, PMR will transfer data to RB.

If subchannels are used (programming in the TSAR), the TSAR will still provide 8 bits data but will activate the TRMASK[7:0] lines. The masked bits are discarded and only the unmasked bits are handled by the protocol machine. In transparent mode, optionally, the masked bits can be replaced with '1' and then transferred to RB.

The CPU can turn off the reception of data for each channel via the command register. When a channel is turned off, a status with 'Frame End (FE)' and 'Receive Abort (RAB)' is transferred to RB (whether or not a frame is open) and the data received from TSAR is discarded until a channel initialization is performed again.

Interrupts are reported by generating interrupt vectors. If several interrupts occur in a very short period of time and could not be reported sequentially (e.g., the internal interrupt bus is busy), these interrupts (maximum of one for each interrupt type) are all stored in the context RAM and reported subsequently at the next possible time.

2.2.1 HDLC mode

Flag	Address	Control	Information	CRC	Flag
0111 1110	8 bits	8 bits	<=0 Bits	16/32 bits	0111 1110

Figure 3
HDLC Frame Format

The frame begin and end synchronization is performed with the flag character (7Eh). Shared opening and closing flag and shared 0 bit between two flags must be supported.

Prior to FCS computation, any '0' bit that directly follows five contiguous '1' bits is discarded. When closing flag is recognized, octet boundary check (bit number divisible by 8), CRC check, and short frame check is performed. The Maximum Frame Length (MFL) check is performed when a data octet is received. If CRC check is disabled, no short frame check is performed. Short frames are discarded in PMR (no transfer to RB). Only a SF interrupt vector is transferred to DMAI. If a CRC, MFL or octet boundary error is detected, data is transferred to RB with a status byte appended. The status byte is also generated when a frame end (closing flag) or an abort sequence (7 or more '1' are received) is detected. The following table describes the format of the word with status byte transferred to RB.

Table 1
Status word

Bit	31	30	29	28	27	26	25	24	23..	..16	15..	..8	7..	..0
Function	FE	MFL	RFO	CRC	ILEN	RAB	–	–	Byte3		Byte2		Byte1	

The number of valid bytes in the status word is indicated via separate signals to the RB.

RAB (Receive Abort), ILEN (Invalid Length), CRC (CRC error), RFO (Receive Frame Overflow), MFL (Maximum Frame Length), and FE (Frame End) bits are status bits transferred transparently through RB to DMAR.

Between two frames, the interframe time fill character can be either 7Eh or Idle pattern (15 contiguous 1's).

If no error, abort or frame end occurs, each time 32 bits are ready, they are transferred to RB. All the data between opening flag and closing flag with or without the CRC value (depending on the channel configuration register) is transferred.

Receive Overflow Error (RFO):

If the transfer to RB is not possible (no free entry in the buffer), RB will activate the RBFULL, and the PMR will then transfer a RFO interrupt (Receive Frame Overflow) to the DMAI and save RFO information in the channel context. The received protocol data (including flags, if any) will be silently discarded. When space is available in RB, the rest of the frame will be transferred and the status bit RFO will be set at the end of the frame, to indicate the software that the frame is not complete and experienced an RB overflow. The interrupt vector is an early warning for the CPU. A possible reaction to the RFO interrupt vector could be to slow down the transmission, so that more FPI and system bus band width is allocated to the receive path. Whenever an error-free frame is transferred to RB, the RFO interrupt, if stored in the context RAM is cleared.

Short Frame Error (SFE):

An SFL-CRC option is provided at the global level to define the lengths of short frame. If the SFL-CRC option is set to 0, then a frame is considered to be a short frame if less than 4 octets for CRC16 or less than 6 octets for CRC32 is received. If the SFL-CRC option is set to 1, then a frame is considered to be a short frame if less than 2 octets for CRC16 or less than 4 octets for CRC32 is received. If CRC check is disabled, then no short frame check is performed.

A flag followed immediately by an abort (01111110-01111111) is considered as a short frame of length 0 bytes. A flag followed by an abort with a shared 0 (01111110-1111111) is also considered as a short frame of length 0 bytes, provided these 7 1's do not form a part of 15 contiguous 1's. Short frames of 0 length is reported even if CRC check is disabled.

Maximum Frame Length Error (MFL):

An MFL option along with the length of MFL (13-bits wide) can be specified. If the MFL option is not set, then the MFL length check is not performed. If the MFL option is set, the frame is considered invalid (MFL error) when the MFL+1st byte is received. This is independent of the CRC mode that is selected. The last bytes of data received depending on the CRC mode is not transferred to RB.

Invalid Length error (ILEN):

Whenever a frame is received containing not an integral number of 8-bit byte data, this error is reported.

CRC Error:

The CRC is checked only if CRC mode is enabled. If CRC error occurs, this is reported via the status information to the RB.

Receive Abort Error (RAB):

When 7 contiguous 1's are received in the middle of a frame, the RAB error is reported via status word unless it is a short frame.

Interrupt: Interframe fill changes

Whenever the interframe fill changes from Flag to Idle Pattern or vice versa, this condition is reported via two interrupts, interframe fill changed to flags and interframe fill changed to Idle Pattern. These Interrupts can be masked on a per channel basis. The interframe fill is considered to be Flags when 2 or more flags (with or without shared 0) is received. The interframe fill is considered to be 'Idle Pattern' if 15 contiguous 1's are received at any time (e.g., after a current frame is aborted or after a closing flag or after MFL error is detected). After reset, the interframe fill is initialized to 'Idle Pattern'.

Interrupt: RFO

The RFO interrupt is generated when RB becomes full while a frame is being received. This interrupt can be masked on a per channel basis.

Interrupt: Short Frame

The Short Frame Interrupt is generated when a short frame is received. This interrupt can be masked on a per channel basis.

The parameters programmed in the channel configuration for HDLC mode are the following:

- Channel On/Off: Channel can be turned on or off using this bit
- CRC mode: 16 bit CRC (1+x5+x12+x16) *or* 32 bit CRC mode (1 + x + x2 + x4 + x5 + x7 + x8 + x10 + x11 + x12 + x16 + x22 + x23 + x26 + x32).
- CRC check suppression: in this case no short frame check is performed and all the data between opening and closing flag is transferred. However, short frames of length 0 bytes will not be transferred to RB independent of this option.
- CRC Transfer: If CRC check is enabled, this option specifies whether or not CRC is transferred to RB.
- Interrupt masks: This indicates whether or not the specific interrupt has to be masked.

Note: no address or control field recognition is performed

2.2.2 Bit synchronous PPP

Flag 0111 1110	Address 1111 1111	Control 0000 0011	Protocol 8/16 bits	Information	Padding	FCS 16/32 bits	Flag 0111 1110

Figure 4
Bit synchronous PPP with HDLC framing structure

Same function in receive operation as in HDLC mode.The same parameters that apply to HDLC also apply to bit-synchronous PPP.

2.2.3 Octet Synchronous PPP

In the octet synchronous PPP mode, an octet stuffing procedure is used instead of the 0 bit insertion/deletion used for HDLC.

The frame begin and end synchronization is performed with the flag character (7Eh). Shared opening and closing flag must be supported but shared 0 bit between two flags is not allowed in this mode.

A table is maintained for each channel containing the Asynchronous Control Character Map (ACCM) for characters 00-1F $_{\text{Hex}}$. One programmable register is used to let the CPU select which of the 32 fixed characters are mapped using the Control Escape sequence. $7D_{\text{Hex}}$ and $7E_{\text{Hex}}$ in the data stream are always mapped.

On reception, prior to FCS computation, each octet with value less than hexadecimal 0x20 is checked in the ACCM. If it is flagged in the receiving ACCM, it is deleted. Each Control Escape octet is also removed, and the following octet is exclusive-or'd with hexadecimal 0x20, unless it is the Flag Sequence (which aborts the frame).

When closing flag is recognized, CRC check, and short frame check is performed. The Maximum Frame Length (MFL) check is performed when a data octet is received. If CRC check is disabled, no short frame check is performed. Short frames are discarded in PMR (no transfer to RB). Only a SF interrupt vector is transferred to DMAI. If a CRC or MFL error occurred, data is transferred to RB with a status byte appended. The status byte is also generated when a frame end (closing flag) or an abort sequence (escape, flag) is detected. Refer to **Table 1** for the format of the word with status byte transferred to RB. Note that an octet boundary check is not performed, because after the opening flag has been detected, the incoming data is handled at the octet level.

Between two frames, the interframe time fill character is always 7Eh.

If no error, abort or frame end occurs, each time 32 bits are ready, they are transferred to RB. All the data between opening flag and closing flag with or without the CRC value (depending on the channel configuration register) is transferred.

RFO Error:

If the transfer to RB is not possible (no free entry in the buffer), RB will activate the RB_FULL, PMR will transfer a RFO interrupt (Receive Frame Overflow) to the DMAI and save RFO in the context of this channel. The received protocol data will be silently discarded. When space is available in RB, the rest of the frame will be transferred and the status bit RFO will be set at the end of the frame, to indicate the software that the frame is not complete and experienced an RB overflow. The interrupt vector is an early warning for the CPU. A possible reaction to the RFO interrupt vector could be to slow down the transmission, so that more FPI and system bus band width is allocated to the receive path. Whenever an error-free frame is transferred to RB, the RFO interrupt, if stored in the context RAM is cleared.

Short Frame Error:

An SFL-CRC option is provided at the global level to define the lengths of short frame. If the SFL-CRC option is set to 0, then a frame is considered to be a short frame if less than 4 octets for CRC16 or less than 6 octets for CRC32 is received. If the SFL-CRC option is set to 1, then a frame is considered to be a short frame if less than 2 octets for CRC16 or less than 4 octets for CRC32 is received. If CRC check is disabled, then no short frame check is performed.

Maximum Frame Length Error:

An MFL option along with the length of MFL (16-bits wide) can be specified. If the MFL option is not set, then the MFL length check is not performed. If the MFL option is set, the frame is considered invalid (MFL error) when the MFL+1st byte is received. This is independent of the CRC mode that is selected. The last bytes of data received depending on the CRC mode is not transferred to RB.

CRC Error:

The CRC is checked only if CRC mode is enabled. If CRC error occurs, this is reported via the status information to the RB.

Receive Abort Error:

When a flag immediately following the control escape octet is received in the middle of a frame, the RAB error is reported via status word unless it is a short frame.

Interrupt: RFO

The RFO interrupt can be masked on a per channel basis.

Interrupt: Short Frame

The Short Frame Interrupt can be masked on a per channel basis.

The parameters programmed in the channel configuration for octet synchronous PPP mode are the following:

- All parameters mentioned for HDLC mode above are applicable.
- An ACCM map of 32-bits per channel specifying what characters in the range 0 to 1F (hex) are mapped.
- An extended ACCM map of 4 characters is maintained at a global level. For each channel, any or all of these 4 characters can be specified to be treated as ACCM character.
- For each channel, an indicator specifying whether or not DEL character should be treated as ACCM character.

2.2.4 Transparent mode

When programmed in transparent mode, the PMR performs fully transparent data reception without HDLC framing, i.e. without

- Flag insertion and deletion
- CRC generation and checking
- bit stuffing.

A programmable character (TFLAG) can be extracted if programmed with FA=1 in the channel specification register. For FA = 0 nothing is extracted.

A programmable sub-channel mode is supported. If subchanneling is not specified (logical channels of less than 64 kbit/s), the TSA_MASK[7:0] lines acts like bit enable lines. The masked bits (TSA_MASK line is low) are replaced with '1' by PMR and transferred together with the unmasked bits to RB after 4 octets have been received from TSAR. In this case FA must be set to '0' for this channel. If subchanneling is specified, only unmasked bits are used to derive the data bits.

To support superchannels (fractional T1 or T1) in the transparent mode, the start of the reception of data is synchronized to the start of a PCM frame (at the framer interface).

Therefore after initialization in transparent mode, the PMR will first stay in an initial state, where all the data received from TSAR are discarded until the TSAR_SYNC line is activated for the first time. The TSAR is responsible to activate the TSAR_SYNC line in the first DS0 timeslot associated to a logical channel composed of more DS0 timeslots.

If the RB is full the received data cannot be transferred. In this case, a status word with status of RFO is sent to RB and a new frame is started only after the next TSAR_SYNC is received.

Interrupt: RFO

The RFO interrupt can be masked on a per channel basis.

The parameters programmed in the channel configuration for transparent mode are the following:

- The extraction of a character with FA bit.
- The character to be extracted in receive operation (TFLAG).
- Subchannel mode: If subchannel is specified, only unmasked bits are used to derive the data bits. Otherwise, the masked bits are replaced with a '1' and handled as if 8-bits of data is received.

2.3 Context and Status RAM / Register (CSR)

The following table describes the context the PMR needs to load and store every time it gets data from TSAR.

Bit / Field Name	Number of Bits	Description
		SEMI-PERMANENT DATA
CH STATUS	1	Channel On/Off state
MODE	2	Protocol Mode (HDLC, Bit PPP, Octet PPP, Transparent Mode)
DISABLE CRC / TFF	1	HDLC/PPP Mode : Disable CRC check Transparent Mode: Transparent Flag Filter
CRC32 MODE / TMP	1	HDLC/PPP Mode : 0=CRC16, 1=CRC32 mode Transparent Mode: Transparent Mode Pack
CRC XFER	1	CRC transfer to RB
INVERT DATA	1	Invert Input Data before processing
E-ACCM	4	Extension ACCM Map in Octet PPP mode
DEL	1	Map the DEL character in Octet PPP mode
INT QUEUE ID	3	Channel Interrupt Queue Identifier
INT MASK - IFF	1	Mask for 'Interframe fill change' Interrupt
INT MASK - SFR	1	Mask for 'Short Frame' Interrupt
INT MASK - RFO	1	Mask for 'Receive Overflow' interrupt
ACCM / TFLAG	32 / 8	Octet PPP Mode: ACCM Map Transparent Mode: Transparent Flag Character
		TRANSIENT DATA
CH STATE	2	Channel State
IFF Indicator	1	Interframe fill last received (Flag or Idle pattern)
RFO	1	RB was full. Status bit appended and transferred to RB when space again available.
SAV ABORT	1.	Abort received but not yet processed
FLAG COUNT	1	Number of flags received

Bit / Field Name	Number of Bits	Description
PENDING INT	4	The interrupts yet to be reported (waiting for Interrupt controller)
CRC	32	Current computed CRC
ONE_COUNT	4	Number of '1' for flag, 0 bit deletion, abort
REM BIT CNT	3	Remainder bit count
REM BITS	7	Remainder bits
NOSYNC	1	Channel in Nosync state (waiting for flag)
FLAG RECD	1	Last Flag Received
BYTE_COUNT	16	Byte count for maximum length check. If exceeded, MFL error bit set.
DATA BUF	56	Current data history (32 in case of CRC32 not transferred plus last 24 bits received)
DATA BUF CNT	3	Number of bytes in DATA BUF

2.4 Interrupt Vector

Following **Table 2** describes the interrupt vector format generated by the PMR.

Table 2
Interrupt Vector Format

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
1	Int Type		0		Queue ID										

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RFO	SF	IF-FL	IF-ID					Channel Number							

Channel number

Identifies the channel where the interrupt occurred.

RFO

Indicates a Receive Frame overflow. The Protocol handler was unable to transfer data to the RB. As soon RB can store data again, a RFO status is appended to the frame.

SF

Indicates that a short frame is detected and discarded by the protocol handler.

IF-FL

Interframe fill changed to Flags. This means that atleast two flags with or without a shared 0 is received as interframe fill.

IF-ID

Interframe fill changed to Idle Pattern. This means that 15 contiguous 1's were received.

Queue ID:

Identifies the queue ID (channel specific).

Int Type:

Type of interrupt is set to 'channel interrupt', since only channel interrupts are generated by PMR.

2.5 Reset Behavior

The PMR can be reset by external HW reset pin or/and by a SW controlled reset.

All internal registers and functions are initialized to known states (RESET state).

After reset the reception of data is turned off for all the channels and all registers are accessible via the FPI slave bus for initialization.

2.6 Functional Test Description

2.7 Production Test Description

3 Macro Interfaces and Signal Description

All signals are active high until otherwise specified. Active low signals are designated by "_N" appended to their names. To make the design as re-usable as possible, a bus signal whose width is application dependent is specified with one of the following parameters:

Parameter name	Bus Type	Typical Value (bits)
		M256F
CN	Channel Number Bus	8
DB	Data/Status Bus	32
OC	Octet Count in Status Word Bus	2

3.1 Signal Description

In the following sections, "Flexible Peripheral Interconnect (FPI) Bus compliant" means that the specified bus uses a subset of the FPI features and satisfies the basic address and data cycle. Not all FPI signals are implemented because default values are sufficient for the application i.e. they can be coded as constants in the hardware. Refer to the FPI bus specification for details of the complete bus.

The following tables lists the FPI-Bus signals and additional out-band signals:

- PRSTAT to indicate if transferred word is status instead of pure data. Captured by RB during address phase.
- PRSTAT_OC to indicate the number of data octets (0 to 3) included in the status word.
- RBFULL to stop PMR transfers when the RB has no free buffer space.

Table 3
Macro Interfaces and Signal Description

Symbol name	I/O	Function
Clock and Reset		
SYSCLK	I	Internal clock (33 MHz) derived from FPI master bus
HW_RESET_N	I	General reset of DMAR. All registers and RAM reset
SW_RESET_N	I	General reset of DMAR. All registers and RAM reset
STOP	I	Stop mode (or test mode).
PR_IIP	O	PR initialization in progress following reset. PR is not availble when this signal is asserted. This signal will remain high for several clocks following release of PR reset(s).
Timeslot Assigner (TSAR) Interface		
TRD[7:0]	I	8 bit data from Time Slot Assigner
TRCH[CNB-1:0]	I	Logical Channel Number (address bus cycle).
TRMASK[7:0]	I	8 bit Mask field to enable single bits of TRD
TRWR_N	I	Request for writing data. Transfer if active.
TRSYNC	I	Logical Channel Synchronization line
PR_TRRDY	O	Protocol data accepted
Receive Buffer (RB) interface		
PR_RBWR_N	O	Only single word write transfer is supported. Transfer if active, no operation otherwise.
PR_RBA[CNB-1:0]	O	Address of bus cycle corresponding to channel number
PR_RBD[DBB-1:0]	O	Output data to be written in the RB
RBRDY	I	Acknowledge signal when PR_RBD is written in RB
PR_RBSTAT	O	Active if Status byte is transmitted in the data word on PR_RBD

Table 3
Macro Interfaces and Signal Description (cont'd)

Symbol name	I/O	Function
PR_RBSTAT_OC [OC-1:0]	O	The number of octets (0 to 3) included in the status word
RBFULL	I	Asserted at end of PMR write cycle if: • Buffer free pool becomes empty or • FIFO task buffer to the DMA channel becomes full. De-asserted after DMAR read cycle.

Interrupt Controller (DMAI) Interface

PR_ICREQ_N	O	Master Request Line
ICGNT_N	I	Master Grant Line
PR_ICD[31:0]	O	Data Bus

FPI Slave Interface

TFPI_A[3:0]	I	Address bus.
PB_TFPI_D[DB-1:0]	I	Data bus. Active during data phase.
TFPI_D[DB-1:0]	I	Daisy Chain data bus input
PR_TFPI_D[DB-1:0]	O	Data bus. Active during data phase.

TFPI_WR_N
TFPI_RD_N
	I
	I
		Read/Write controls. Following codes are defined:
		WR_N = 1; RD_N = 1 => NOP
		WR_N = 0; RD_N = 1 => data written to PMR
		WR_N = 1; RD_N = 0 => data read from PMR

TFPI_VG_SEL_N	I	Virtual Global Select. Global data for PMR (e.g., MFL) is updated using this select signal.
TFPI_VC_SEL_N	I	Virtual Channel Select. Channel specific data (e.g., disable CRC) is updated using this select signal.
TFPI_PR_SEL_N	I	PMR Select. PMR macro specific data is updated using this select signal.
PB_TFPI_RDY	I	Ready enable. The select signal from TFPI is processed only when this signal is active.
PR_TFPI_RDY	O	PMR action completed

3.2 Data Flow and Functional Timing

3.2.1 FPI Slave Bus

Refer to the SIEMENS FPI Bus for Munich-Macros Specification.

3.2.2 Timeslot Assigner (TSAR) Interface

The transfer is started by TSAR with the TRWR_N line. In the same cycle the channel number is activated. In the next cycle, the data and mask line is transferred.

For the first active timeslot of each logical channel, the TSAR activates the TRSYNC line.

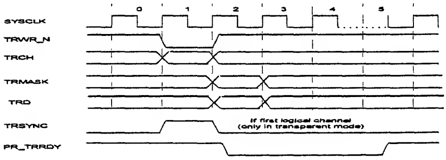


Figure 5
Data transfer from TSAR to PMR

3.2.3 RB Interface

The RB interface to the PMR is based on a unidirectional FPI slave bus. Only two operation codes, NOP and Single Word Transfer, are defined. The write signal WR_N is used to indicate a single word tranfer to RB from PMR. The select signal SEL_N is implicitly active and not physically present. The PMR initiates an address cycle by asserting the WR_N. RB captures the channel number from the address bus during this cycle. During the data cycle, RB captures the data as soon as possible. RB asserts RBRDY during the clock cycle that it can complete the data transfer.

Following out-of-band signals are required

1.PMR_STAT to indicate if transferred word is status instead of pure data. Captured by RB during address phase.

2.PRSTAT_OC to indicate the number of data octets (0 to 3) included in the status word

3.RBFULL to stop PMR transfers when the RB has no free buffer space.

The PMR bus does not support overlapped transfers. The RBFULL signal is asserted simultaneously with RBRDY. PMR must not attempt another transfer until this flag is de-asserted.

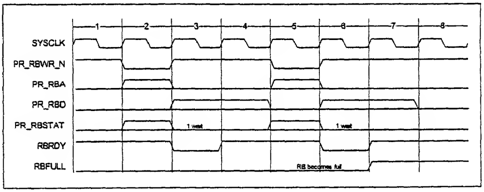


Figure 6
Two word transfer from PMR to RB

3.2.4 DMAI Interface

The interrupt vector is transferred to the DMUI on the interrupt bus. PMR asserts the PR_ICREQ_N and waits for the ICGNT_N grant. When the grant is received, PMR removes the request at the beginning of the next clock cycle and asserts the PMR interrupt vector. No other handshaking is required.

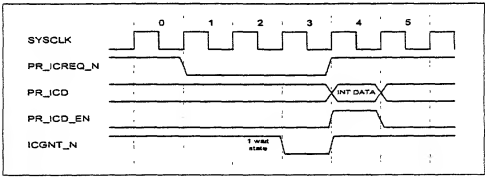


Figure 7
Two Interrupt transfers to DMAI

3.3 Macro Functional Test

3.4 Macro Production Test

4 Register Description

4.1 Register Overview

The PMR specific registers allows initialization and turn off of the protocol handler for each channel.

4.2 Detailed Register Description

4.2.1 Configuration Register 1 (CONF1)

Access	: read/write
Offset Address	: 00_H
Reset Value	: 83000060_H

31GC						GC	GC				
IIP (read only)	0	0	0	0	0	STOP	SW_RESET	0	0	MFLE	MFL(12:8)

15								
MFL(7:0)	MBIM	PBIM	RBIM	0	SFLCRC	RBM	LBE	GPM

Note: This register is implemented once in the chip !

GPM: General PCM Port Mode:
'0' : Standard Channelized Mode;
'1': Alternate Channelized Mode: 8 * T1/E1 lines (port 0...7) without chip internal framing function

LBE: Little/Big Endian Byte Swap: '0': Little Endian ; '1' : Big Endian

RBM: Receive Buffer Monitor: if set to '0', the minimum free pool count is captured in register RBTH; if set to '1' an interrupt is generated, if the free pool counter falls blow the value programmed in register RBTH

SFLCRC: '0': short frame is detected, if a received frame contains < 4 bytes (CRC16)

or < 6 bytes (CRC32)
'1': short frame is detected, if a received frame contains < 2 bytes (CRC16) or < 4bytes (CRC32)

RBIM: Receive Buffer Interrupt Mask: if set to '1' the receive buffer threshold system interrupts (RAEW,RBEW) are not generated

PBIM: PCI Interface Interrupt Mask: if set to '1' the PCI Interface interrupt is not generated.

MBIM: Mailbox Interrupt Mask: if set to '1' the Mailbox interrupt to the PCI interface is not generated

MFL(12:0): Maximum Frame Length check against this value is performed, if MFLE ='1'

MFLE: The maximum frame length check is enabled (='1') or disabled (='0')

SW_RESET: if set to '1', the software reset is activated; software reset has generally the same effect on the HDLC part as hardware reset with two exceptions:
- outputs are not tristated
- the bit SW_RESET itself is not reset; it has to be deactivated by software again

STOP: '1' means : device is in "Fast Software Initialisation Mode" (for customers only applicable after reset!); '0' : device is in normal operation

IIP: Initialization in Progress (Read Only): If '1', internal RAMs are being self initialized by the device; no other access than to this register is possible

Note: Only the following fields are handled by PMR:
SFLCRC,
MFL
MFLE

4.2.2 (Virtual) Channel Specification Command (CSPEC_CMD)

Access	: write
Address	: 20_H
Reset Value	: 00000000_H

31	16
CMD_XMIT(7:0)	CMD_REC(&:0)

15 8	7 0
00	CHAN(7:0)

Note: The Virtual Channel Spec (VCS) Command Register has to be programmed after all other required VCS registers, in order to initiate the programming of all macros. In case of a debug command, first the command register has to be written, then a broadcast read of the virtual channelspec is possible by read of all VCS data registers. Only the macro which has implemented the corresponding bit has to drive the register bit, otherwise drives '0' to provide broadcast read feature.

Note: ONLY THE RECEIVE COMMANDS ARE HANDLED BY PMR.

CHAN(7:0): selected channel number to be programmed

CMD_REC(7:0): for details refer to table "Command Description"

Commands Receive:

1. Receive Init
2. Receive Abort
3. Receive Off
4. Receive Hold Reset
5. Receive Debug
6. Receive NOP

CMD_XMIT(7:0): for details refer to table "Command Description"

Commands Transmit:

1. Transmit Init (sets up a TBUFFER= ITBS automatically)
2. Transmit Abort
3. Transmit Off; (shuts down the TBUFFER to NITBS=0 automatically (regardless of NITBS register contents)
4. Transmit Hold Reset
5. Transmit Debug
6. Transmit NOP (no operation)
7. Transmit Idle (Sends IDLE code as programmed in CONF3 on the channel)

Note: Transmit Init for a channel must be programmed only after reset or after a Transmit Off command, i.e. two Transmit Init commands for the same channel are not allowed

Command Table Receive:

23	22	21	20	19	18	17	16	
Rsvd	Rsvd	Rsvd	Debug	HOLD RESET	ABORT	OFF	INIT	function
0	0	0	0	0	0	0	1	receive init
0	0	0	0	0	0	1	0	receive off
0	0	0	0	0	1	0	0	receive abort
0	0	0	0	1	0	0	0	receive hold reset
0	0	0	1	0	0	0	0	receive debug
0	0	0	0	0	0	0	0	receive nop

Command Table Transmit:

31	30	29	28	27	26	25	24	
Rsvd	Rsvd	Rsvd	Debug	HOLD RESET	ABORT	OFF	INIT	function
0	0	1	0	0	0	0	0	transmit idle
0	0	0	0	0	0	0	1	transmit init

31	30	29	28	27	26	25	24	
0	0	0	0	0	0	1	0	transmit off
0	0	0	0	0	1	0	0	transmit abort
0	0	0	0	1	0	0	0	transmit hold reset
0	0	0	1	0	0	0	0	transmit debug
0	0	0	0	0	0	0	0	transmit nop

4.2.3 (Virtual) Channel Specification Mode Receive (CSPEC_MODE_REC)

Access	: read/write
Address	: 24_H
Reset Value	: 00000000_H

31															
			DEL	ACCMX(3:0)				TFLAG(7:0)							

15															0
0	0	TFF	INV	TMP	CRC XFER	CRC 32	CRC DIS	0	0	0	0	0	0	PMD(1:0)	

PMD(1:0): Protocol Machine Mode

CRCDIS: '0': CRC check is enabled; '1': CRC check is disabled

CRC32: if set to '1', add CRC32 checksum; otherwise add CRC16 checksum

CRCXFER: '0': CRC is not transfered to the shared memory; '1' CRC is transferred to the shared memory; if CRCXFER is selected the short frame check is not applied to the received data

TMP: Transparent Mode Pack: '1': if subchanneling is used in transparent mode (i.e. less than 8 bits of a time slot are used), the non-used (masked) data bits are discarded;

'0': the non-used (masked) data bits are substituted by '1's

INV: '1' all received data in this channel are inverted; '1': no inversion

TFF: function identical to FA in transmit direction (only transparent mode)

TFLAG(7:0): used only in transparent mode; these bits constitute the flag that is filtered from the received bit stream

ACCMX(3:0): Extended ACCM: a '1' enables the corresponding character in CSPEC_REC_ACCMX

DEL: Del Map flag

4.2.4 (Virtual) Channel Specification Receive ACCM Map (CSPEC_REC_ACCM)

Access	: read/write
Address	: 28_H
Reset Value	: 00000000_H

31															16
1F	1E	1D	1C	1B	1A	19	18	17	16	15	14	13	12	11	10

15															0
0F	0E	0D	0C	0B	0A	09	08	07	06	05	04	03	02	01	00

This map is used by a channel in octet synchronous PPP mode only. If a bit is set, the corresponding character is expected to be mapped by the Control ESC character and is hence ignored, if received.

4.2.5 Receive Extended ACCM Map (REC_ACCMX)

Access	: read/write
Address	: $2C_H$
Reset Value	: 00000000_H

31	16
CHAR3(7.0)	CHAR2(7:0)

15	0
CHAR1(7:0)	CHAR0(7:0)

This register is only used by a channel in octet synchronous PPP mode. A character written to this register will be mapped with a Control Escape sequence, if the corresponding enable flag is set in the CSPEC_MODE_REC register (ACCMX(3:0)).

4.2.6 (Virtual) Channel Specification Buffer (CSPEC_BUFFER)

Access	: read/write
Address	: $3C_H$
Reset Value	: 00000000_H

31

TTC(2:0)	0	ITBS(11:0)

15 0

0	TQUEUE(2:0)	0	TBTC(2:0)	0	RQUEUE(2:0)	0	RBTC(2:0)

RBTC: receive burst threshold code (affects RB)

RQUEUE: The generated receive interrupts will be sent to this interrupt queue

TBTC: transmit burst threshold code (affects TB)

TQUEUE: The generated transmit interrupts will be sent to this interrupt queue

ITBS: individual transmit buffer size (affects TB);

TTC: Transmit Threshold Code

Note: Only the Receive Interrupt Queue (RQUEUE) is processed by PMR.

4.2.7 Test Command Register (TAC)

Access	: write
Offset Address	: FE_H
Reset Value	: 00000000_H

31					
MID	0	0	0	AI	CMD

15				0
0	0	0	0	ADDRESS

MID:	Macro ID Code
AI:	Auto Increment Function
Address:	internal address
CMD:	Command (select of ram and register)

CMD:

23	22	21	20	19	18	17	16	
								function
0	0	0	0	0	0	0	0	tbd (macro spec)

Macro ID Code:

31	30	29	28	macro
0	0	0	0	PMR
0	0	0	1	PMT
0	0	1	0	RB

31	30	29	28	macro
0	0	1	1	TB
0	1	0	0	DMUR
0	1	0	1	DMUT
0	1	1	0	TSAR
0	1	1	1	TSAT
1	0	0	0	XPI
1	0	0	1	DMUI

General

- the test access provides read/write access of important internal rams and registers
- test registers are virtuals global registers (SEL signal: pb_vg_tfpi_sel_n / implemented as a daisy chain).
- the single macros are selected by the MID code of the test command register
- the test access provides read/write access of important internal rams and registers
- CMD specifies/selects one of the macro rams/registers
- the address field is used to access a ram address
- AI: autoincrement : address given in the address field is incremented automatically for each access
- All macros which are not selected by MID drive data output "00000000" and <macro>_TPFI_RDY = '1'. Driving "00000000" would mean not disable the enable line for data out, but to set the output data to "00000000".

Test write access:

1. Write TAC
2. Write TAD

Test read access:

1. Write TAC
2. Read TAD

Typically the macro selected via MID delays the RDY signal until the selected ram/ register has been read and the data can be provided at the TFPI interface. No prefetch of testdata is required.

CMD:

0: Configuration RAM 0- all data excluding ACCM

1: Configuration RAM 1- ACCM

2: Transient RAM 2- CRC

3: Transient RAM 3- data buffer word (first 4 bytes of data buffer)

4: Transient RAM 4- data buffer remaining 3 bytes, data buffer count

5: Transient RAM 5 - destuffer context, Frame Length

6: Transient RAM 6 - channel state context, save abort, interrupts, interframe fill state, flag count

7: Stop PMR

8: Continue PMR

4.2.8 Test Data Register (TD)

Access	: read/write
Address	: FF_H
Reset Value	: 00000000_H

31
TEST DATA

15	0
TEST DATA	

TEST DATA: ram/register test data (read or write)

Note: It is assumed that a TFPI_SEL_N signal is generated for register access of common TAC and TD register. For write / read access the MID (macro ID) is used for addressing one macro: If MID does not match with the hard coded ID, TFPI_D are prepared corresponding the 'daisy chain' requirements.I.e. each macro drives "00000000". These '0' s are ored in the macroshell.

4.2.9 (Virtual) Channel Specification Mode Transmit (CSPEC_MODE_XMIT)

Access	: read/write
Address	: 30_H
Reset Value	: 00000000_H

31					
0	0	0	DEL	ACCMX(3:0)	TFLAG(7:0)

15														0
IFTF	SFE	FA	INV	TMP	0	CRC 32	CRC DIS	0	0	0	0	0	0	PMD(1:0)

PMD(1:0): Protocol Machine Mode

CRCDIS: '0': CRC is transmitted; '1': CRC is not transmitted

CRC32: if set to '1', add CRC32 checksum; otherwise add CRC16 checksum

TMP: Transparent Mode Pack: '1': if subchanneling is used in transparent mode, the adjustment of transmit data is done automatically;
'0': the non-used bits are expected to be '1's in the shared memory

INV: '1' all transmit data in this channel are inverted; '1': no inversion

FA: flag adjustment (only transparent mode) selected

SFE: Shared Flag Enable: '0' 2 flags are sent at least between (HDLC, PPP?) frames; '1' flag is sent at least between (HDLC,PPP?) frames

IFTF: Interframe Time Fill: '0': $7E_H$; '1': FF_H

TFLAG(7:0): used only in transparent mode; these bits constitute the flag that is inserted into the transmit bit stream

ACCMX(3:0): Extended ACCM:

DEL: Del Map flag

4.2.10 (Virtual) Channel Specification Interrupt Mask/Queue (CSPEC_IMASK)

Access	: read/write
Address	: 48_H
Reset Value	: $FFFFFFFF_H$

31 Transmit

HI	HTAB	0	TAB	0	0	0	0	UE	FE	0	0	0	0	CMDF	CMDC

15 Receive 0

HI	HRAB	FE	RAB	MFL	ROFD	CRC	ILEN	RFOP	SF	IFFL	IFID	0	SD	0	CMDC

Interrupt Generation Mask: These bits correspond to the respective channel interrupts, if set to '1', the corresponding interrupt will not be generated by the device

SD : Silent Discard Mask: If set to '1' the HRAB+RAB interrupt, which occurs, when receive frames are discarded during HOLD is masked

5 Functional Test Specification (Macrolevel)

(use Checklist C06 "Function Checklist - Comparison of Specification vs. Circuit" http://www.hl.siemens.de/lognet/hl_ta/dhb/f.htm)

Appendix L

Title:
Timeslot Assigner Receive V2.2.1

1 Introduction

1.1 Overview

The Timeslot Assigner Receive TSAR (macro name key: TR_) maps the incoming receive timeslots (maximal 32 ports x 32 timeslots = 1024 timeslots and for the M256F 28 ports x 24 timeslots = 672 timeslots)to logical channels and provides the channel relevant information to the protocol machine interface.

1.2 Features

- coordinates requests of 32 ports [M256F: 28ports] by built-in arbitration
- arbitration priority from low (port n = 0) to high (port n=31)
- channel programming by indirect access through TFPI (FPI Slave) Interface
- data output FIFO buffering for adjustment of further data processing
- programmable channel assignment per timeslot
- programmable mask per timeslot
- programmable timeslot inhibit flag per timeslot
- remote channelwise loop, one channel at a time (loop RD -> TSAR -> TSAT -> TD)
- remote portwise loop, one port at a time (loop RD --> TSAR --> TSAT --> TD)
- loop supported by a jitter attenuator, consisting of a 512 bit FIFO with slip function
- programmable "TMA1ST flag" (tmafirst flag) to identify the 'first' timeslot for TMA mode channels
- FPI target interface
- performance: SYSCLK up to 70 MHz
- number of gates:
- area:
- power consumption:
- full scan path
- RAM (Timeslot Assigner Receive Parameter Table: TARPT) built in selftest
- RAM (Timeslot Assigner Receive Loop Fifo: TARLPFIFO) built in selftest
- RAM (Timeslot Assigner Receive Data Fifo: TARFIFO) built in selftest

1.3 System Integration

The TSAR (macro name key: TR_) has four interfaces:

- one FPI Slave (TFPI) Bus
- parallel REQ/GNT interface to (R)XPI functions (accept incoming data and corresponding timeslot information)

- REQ/GNT interface to PMR functions (provides channel information and data to processing functions)
- altemate output for remote loop (for one channel or port at a time)

The TSAR receives the synchronized data and corresponding port and time slot information from the (R)XPI block. The incoming data is mapped to channel relevant information and finally provided to the protocol machine. Additionally this data can be selected for one single channel or port to provide the loop interface. Configuration of the TSAR is done via the FPI slave interface.

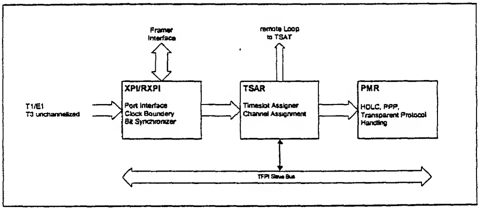


Figure 1.3:

System integration

1.4 Known Restrictions

1) necessary minimum SYSCLK to serve 32 ports at E1 data rate (32 timeslots), without regarding synchronization losses

one access request per timeslot; i.e. 2.048Mbit x 32 ports / 8bits/timeslot = 8 Mbyte/s and 4 clock cycles for processing (arbitration, mapping, data transfer) 33MHz SYSCLK would be sufficient.

in m256f application, i.e T1 1.544Mbit x 28ports / 8bits/timeslot = 5.4 Mbyte/s and 4 clock cycles for processing, the minimum frequency for SYSCLK = 22 MHz.

2) priority and waitstates for indirect read/write access from the FPI slave interface to the TARPT RAM.

For operation with maximum data rate, data processing takes every forth clock cycle access to the TARPT RAM. Else the TFPI slave interface has access to the TARPT. Thus an indirect write access can be delayed 1 clock cycle for write and 3 clock cycles for read.

3) restrictions regarding the remote loop:

a) Channelwise loop: same port for receive and transmit direction, same number of timeslots activated (inhibit bit = 0), identical mask bit field for all timeslots that are activated

b) Portwise loop: same port for receive and transmit direction, only one channel is supported which comprises all timeslots, inhibit bits are not allowed, identical mask bit field for all timeslots

4) timeslot to channel assignment programmed in the parameter table for unchannelized mode .

for unchannelized mode all timeslots of a port have to be assigned to one channel,. valid timeslots are timeslot 0 to 23.

2 Functional Description

2.1 Block Diagram

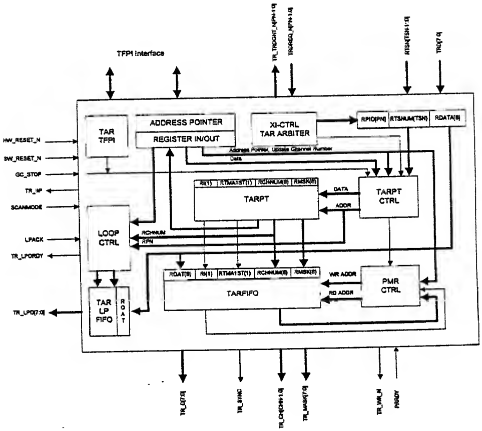


Figure 2.1:
TSAR Block Diagram

2.2 Normal Operation Description / Reset

With an active reset (HW_RESET_N or SW_RESET_N) all registers are set to their benign state and the TSARFIFO is reset to empty (there is no difference between HW and SW reset). After the reset signal has become inactive, the self initialization procedure in the TARPT RAM shell starts resetting the RAM values to RI = '1' (inhibit timeslot), TMA1ST = '0' (no TMA sync timeslot), CHNUM=00hex (channel number 0) and MASK = 00hex (all bits masked).

During this initialization period the TSAR is held in stop mode. The TSAR is inaccessible from all interfaces and doesn't generate any request itself. The active self init period is indicated by output TR_IIP = '1', which becomes '0', when the RAM TARPT is initialized.

For fast initialization of the TARPT RAM an input GC_STOP is still asserted (reset value) and keeps the TSAR in stop mode. In stop mode the contol logic assigns the TARPT access exclusively to the TFPI interface and normal operation to this RAM is suspended. Thus the initial programming of the TARPT can be achieved very fast since it is not interleaved by normal operation accesses. A request from the (R)XPI will be processed as in normal operation mode, but the timeslot and data information is discarded. The TARPT is programmed now by writing to the two TSAR indirect access registers. The TARPT address corresponds to the time slot number and port number.

In normal operation mode (GC_STOP='0') TFPI re-programming requests to TARPT are handled by the control logic as ordinary requests for access. Thus they can be delayed up to 1 clock cycles for write until the access is granted. For read cycles valid data is delayed 2 additional clock cycles because the access to the TARPT is pipelined. Before reprogramming a channel to timeslot mapping configuration the corresponding channel must be turned off in PMR, in order to avoid intermediate TARPT states for timeslots of that channel. The TARPT contains the following entries per time slot (address):

TARPT.RI (inhibit receive time slot)

TARPT.RTMA1ST (flag to be used by PMR)

TARPT.RCHNUM[7:0] (channel number for that timeslot)

TARPT.RMASK[7:0] (mask bits for that timeslot)

After deasserting the GC_STOP input signal, the TSAR is in normal operation mode. The arbiter now accepts requests from the TRDREQ_N signals and grants the data bus TRD and the time slot number bus RTSN to the selected initiator by asserting the corresponding TR_TRDGNT_N signal. After deasserting the corresponding TRDREQ_N signal (marks valid data on the busses) data is sampled for internal processing.

The following information is received from the (R)XPI:

requests for data processing from the (R)XPI : TRDREQ_N[PN-1:0]
timeslot number of granted (R)XPI : RTSN[4:0]
receive data of granted (R)XPI: TRD[7:0]

The timeslot number, which is received on RTSN bus is combined with the port ID, that the arbiter generates from the corresponding request line. The concatenation of timeslot number and port id points to a location in the TARPT. The corresponding contents is read and written into the TARFIFO. A transfer to PMR is started by TSAR by activating the request line TR_WR_N. In the same cycle the channel number and the mask lines are activated. For the first active timeslot of each logical channel in TMA mode, the TSAR activates the TR_SYNC line. This TMA1ST info flag is used in the protocol machine receive for synchronisation purposes to indicate the first timeslot in a frame, which is assigned to a multi-timeslot TMA channel. It must also be set for a single timeslot TMA channel. The data transaction is finished by PMR when the PRRDY line is active.

The following information is delivered to the PMR:
the receive data, which were received at TRD[7:0]: TR_D[7:0]
the assigned channel number: TR_CH[7:0]
the programmed mask for that timeslot: TR_MASK[7:0]
the TMA1ST info flag: TR_SYNC

2.3 Remote Loop

A basic channelwise or portwise remote loop is available from the TSAR to the TSAT. The payload of one (and only one) logical channel or port is mirrored from the receive part to the transmit part of the looped port. In SCM mode, the framing bits, CRC and spare bits are not looped. They are provided by the M256F framer and the facility data link controller. The channel number or port ID is programmed in the configuration register 2 CONF2. The loop is activated by setting the corresponding loop command bit in the configuration register 2 CONF2. Receive timeslots of that channel or port matching the programmed channel number or port ID are written to the TSAR loop FIFO (64 x 1 byte) which is implemented as a circularly organized memory controlled by a read pointer and a write pointer. The loop FIFO acts as a jitter attenuator which compensates the clock jitter between receive and transmit clock.

Only one loop command bit, either for channel or port loop, should be set at a time. Before programming the channel number or port ID, the loop must be turned off. Generally, the port number or channel number has to be programmed first before setting the portwise or channelwise loop command bit.

In case of a channel loop, the channel characteristics in receive and transmit direction must be identical. The numbers of active receive and transmit timeslots of the looped channel must be identical. The mask bit fields in the active timeslots must be identical.

In case of a port loop, only one channel is allowed which comprises all timeslots. Inhibit bits for the looped port are not allowed.

The data transfer for the looped channel or port to the PMR is not affected The loop is turned off by resetting the corresponding loop command bit. The LPDRDY signal is then deactivated, and the loop FIFO will be reset to empty.

Slip Function

After activating the loop, 32 receive data bytes are written to the loop FIFO until the LPDRDY signal is asserted by TSAR. TSAT then starts reading the transmit data from the loop FIFO. Hence, the initial distance between the FIFO read pointer (RP) and the write pointer (WP) is 32 bytes. Due to a RCLK/TCLK clock jitter the RP may move towards WP. In case the distance between RP and WP is equal plus/minus 1 byte a slip of the read pointer will occur. Provided RP is faster than WP a positive slip occurs (the previously received 31 bytes are read out twice). In case RP is slower than WP a negative slip occurs (the next 31 received bytes are skipped). The slip condition is checked with each access to the loop FIFO.

3 Macro Interfaces and Signal Description

All signals are active high until otherwise specified. Active low signals are designated by "_N" appended to their names. To make the design as re-usable as possible, a bus signal whose width is application dependent is specified with one of the following parameters:

Parameter name	Bus Type	Typical Value (Bits) M256F
PN	max. port number bus	5
TSN	max. timeslot number bus	5
CHN	max. channel number bus	8
DB	data bus width	32
AB	address bus witdh	30

3.1 Signal Description

3.1.1 Global Signals

Signal Name	Direction	Type	Tsu/ Thld Td	meaning/comment special characteristics
SYSCLK	i			internal system clock
HW_RESET_N SW_RESET_N	i i			general SW and HW Reset
SCANMODE	i			SCAN Test mode
GC_STOP	i			external init keeps TSAR in stop mode, exclusive access to TARPT RAM from TFPI
TR_IIP	o			TSAR initialization in progress following reset. TSAR is not availble when this signal is asserted. This signal will remain high for several clocks following release of TSAR reset(s).

3.1.2 Receive Port Interface

The XPI(RXPI) interface consists of the following signals

Signal Name	Direction	Type	Tsu/ Thld Td	meaning/comment special characteristics
TR_TRDGNT_N[PN-1:0]	o			data/tsnum bus is granted to receive port n=index
TRDREQ_N[PN-1:0]	i			request from receive port n=index for servicing/ reading 8 bit data
TRD[7:0]	i			data bus containing 8 bit serial/parallel converted time slot data (bit 0 is the first serial bit received, bit 7 is the last serial bit received)
RTSN[TSN-1:0]	i			time slot bus containing the time slot number of the corresponding data XI_TRD (valid time slot numbers are 0 to 23 for T1, 0 to 31 for E1 and **for unchannelized mode all timeslots of a port have to be assigned to one channel**

Description of the (R)XPI interface protocol:

As soon as any port RXPI[n] has 8 bit data available for processing, it asserts the signal TRDREQ_N to the arbiter part of the TSAR, in order to request service. The arbiter then grants by activating the respective grant signal TR_TRDGNT_N one port access to the busses TRD (data) and RTSN (time slot number). TRDREQ_N is deasserted to indicate valid data on the busses TRD and RTSN. TSAR reads the bus information and deasserts the TR_TRDGNT_N.

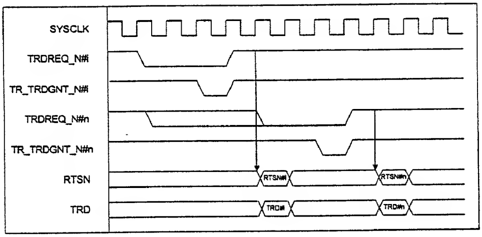


Figure 3.1.2:
Data Transfer from XPI (RXPI) to TSAR

3.1.3 Protocol Machine Receive Interface

The PMR interface consists of the following signals

Signal Name	Direction	Type	Tsu/Thld Td	meaning/comment special characteristics
TR_WR_N	o			request for servicing/writing data
TR_D[7:0]	o			8 bit data for PMR
TR_CH[CHN-1:0]	o			logical channel number
TR_MASK[7:0]	o			8 bit mask field to enable single bits of TSA data
TR_SYNC	o			logical channel synchronization line in TMA mode
PRRDY	i			PMR finished data transaction

Description of PMR interface protocol:

Whenever data in the TARFIFO is available a transfer is started by TSAR asserting the signal TR_WR_N. In the same cycle channel number, data and mask information is activated. For the first active timeslot of each logical channel in TMA mode, the TSAR also actives the TR_SYNC line. The information is held stable until PMR finished data transaction by activating the line PRRDY.

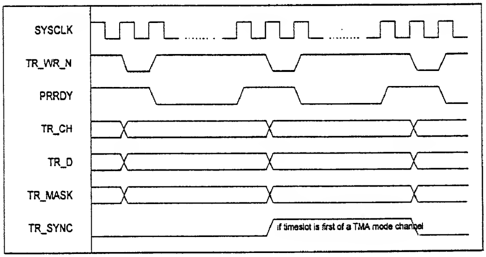


Figure 3.1.3:
Data transfer from TSAR to PMR

3.1.4 **TSA Transmit Interface** (for remote loop)

TheTSA transmit interface consists of the following signals

Signal Name	**Dir ecti on**	**Type**	**Tsu/ Thld Td**	**meaning/comment special characteristics**
LPACK	i			acknowledge from TSAT data has been read
TR_LPD[7:0]	o			8 bit timeslot data of channel or port which is in remote loop operation
TR_LPDRDY	o			indicates valid data available for loop operation

Description of TSA transmit interface protocol:

The remote loop is activated by programming a remote loop with loop active flag set (CRLP or RLP). The corresponding timeslot data and mask bit field of the channel/port matching with programmed channel number (LPCN) or port ID (LPPID) is written into the

TAR LPFIFO. As soon as the LPFIFO is filled up, TR_LPDRDY signal is activated indicating valid data for the remote loop. The data is read out from the TAR LPFIFO by activating the signal LPACK. The remote loop is deactived by resetting the loop active flag.

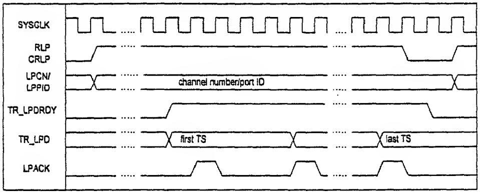


Figure 3.1.4:
Data transfer on the loop interface

3.1.5 FPI Slave Interface

In the following sections, "Flexible Peripheral Interconnect (FPI) Bus compliant" means that the specified bus uses a subset of the FPI features and satisfies the basic address and data cycle. Not all FPI signals are implemented because default values are sufficient for the application i.e. they can be coded as constants in the hardware. Refer to the FPI bus specification and FPI + Target Template Bus for details of the complete bus.

The TFPI interface consists of the following signals:

Signal Name	Direction	Type	Tsu/ Thld Td	meaning/comment special characteristics
TFPI_RD_N	i			Read control input
TPFI_WR_N	i			Write control input
TFPI_A[AB-1:2]	i			Address input
TFPI_D[DB-1·0]	i			Data input
TFPI_RDY	i			Ready input
VG_TFPI_SEL_N	i			(Macro) select input (broadcast programming virtual global register)
TFPI_SEL_N	i			(Macro) select input (macro specific programming)
TR_TFPI_D[DB-1:0]	o			Data output
TR_TFPI_D_EN	o			Data enable (active high)
TFPI_RDY	o			Ready output
TR_TFPI_RDY_EN	o			Ready enable

4 Register Description

4.1 Register Overview

Register Overview Table : TSAR V2.1

Register ID	Access	Absolute Address cs_n & a(7:2)	Reset Value	Comment
CONF2	R/W	44_H	00000000_H	(virtual global) configuration register 2
TSAIA	R/W	70_H	00000000_H	timeslot assignment indirect access register
TSAD	R/W	74_H	02000000_H	timeslot assignment data register
TAC	R/W	58_H	00000000_H	(virtual global) test command register
TD	R/W	$5C_H$	00000000_H	(virtual global) test data register

4.2 Detailed Register Description

4.3 (Virtual Global) Configuration Register 2 (CONF2)

Access	: read/write
Address	: 00000044_H
Reset Value	: 00000000_H

31															16

15	14		12 8	7 0
CRLP	RLP		LPPID[4:0]	LPCN[7:0]

Note: *This register is common for TSAT and other macros. The macro TSAT just collect those data bits from the written dword (see above), that are relevant for macro TSAT. With a read access to this register, the macro TSAT drive only a value for the relevant bits. All other bits are set to '0'.*

LPCN[7:0]:	Channel selection for channelwise remote loop
LPPID[4:0]:	Port selection for portwise remote loop
CRLP:	'1'--> channelwise remote loop on channel selected by LPCN[7:0]
RLP:	'1' --> portwise remote loop on port selected by LPPID[4:0]

4.3.1 Timeslot Assignment Indirect Access Register (TSAIA)

Access	: read/write
Address	: 00000070_H
Reset Value	: 00000000_H

31								23							16
DIR	0	0	0	0	0	0	0	AI	0	0	0	0	0	0	0

15				7			0
0	0	0	PORT[4:0]	0	0	0	TSNUM[4:0]

Address:

This register specifies the base address for access to the TARPT (TSAR parameter table RAM). The address is composed of timeslot number and port id. The address bit field TSNUM[4:0] corresponds to the timeslot number of port which is selected by the address bit field PORT[4:0].

TSNUM[4:0]	timeslot number which is affected by access (T1 = 0..23, E1 = 0..31)
PORT[4:0]	port number which is affected by access (port 0..31)

DIR: transmit/receive direction

1--> Timeslot Assigner Transmit

0--> Timeslot Assigner Receive

AI: auto increment mode

If set when writing a start address to the address pointer, the TSNUM bit field of the address will be automatically incremented (and wrapped at the maximum value x1F) after each read or write access to the data register.This simplifies the transfer of data blocks because the address register for a certain port has to be written only once at the beginning of a data transfer.

Read access to TSAIA

On read access to TSAIA TSAR responds, when DIR is set to '0' (which is the reset value). When DIR is set to '1' (which needs a write access first in order to set DIR) the TSAT should respond. The macro, which is not accessed, should respond with all '0'.

In case having used the autoincrement mechanism before, the returned value should be the actual address, i.e. the autoincremented timeslot number.

4.3.2 Timeslot Assignment Data Register (TSAD)

Access : write

Address : 00000074_H

Reset Value : 02000000_H

31						25	24								16
0	0	0	0	0	0	RTI	RTM A1ST	0	0	0	0	0	0	0	0

15	0
RCHNUM[7:0]	RMASK[7:0]

The data register is a port into the TARPT RAM for read and write access. The RAM address is specified by the address bit field in the timeslot assignment indirect access register. The TARPT RAM is initialized following hw_reset_n and sw_reset_n.

RMASK[7:0] receive mask bit field for bit rate adaption, mask bit = '1' bit is enabled, mask bit = '0' bit is discarded (reset value = 00_H)

RCHNUM[7:0] receive channel number, maps the corresponding timeslot to this logical channel (reset value = 00_H)

RTMA1ST receive TMA first (for TMA synchronisation purpose), TMA = '1' indicates first timeslot in a logical TMA channel (reset value = '0')

RTI receive timeslot inhibit, RTI = '1' no request to the protocol machine for this timeslot, timeslot is discarded (reset value = '1')

4.3.3 (Virtual Global) Test Command Register (TAC)

Access	:read / write
Address	: 00000058_H
Reset Value	: 00000000_H

31			25	24	23 16
MID	0	0	0	AI	COMMAND

15				0
0	0	0	0	ADDRESS

Refer to the Implementation Specification M256F for details to the TAC register.

MID:	Macro ID Code (**TSAR: "0110"**)
AI:	auto increment function
ADDRESS:	internal address
CMD:	command (select of RAM or register)

CMD:

Specifies the access to memory block or register being addressed. Defined command types are :

00000001 : TARPT testmode read (TSAR parameter table RAM)
10000001 : TARPT testmode write

On read access to TAC the macro which is selected via MID should respond to this access. All other macros should respond with all '0'.

4.3.4 (Virtual Global) Test Data Register (TD)

Access	: read/write
Address	: $0000005C_H$
Reset Value	: 00000000_H

31	16
TEST DATA	

15	0
TEST DATA	

Refer to the Implementation Specification M256F for details to the TD register.

The TD register is a port into theTARPT (TSAR parameter table RAM) for read and write access. The select of RAM and access type is specified by the command type bit field in the register TAC. The RAM address itself is specified by the address bit field in the register TAC.

TARPT: TSA receive parameter table RAM

TEST DATA[7:0]:	receive mask bit field
TEST DATA[15:8]:	receive channel number
TEST DATA[16]:	receive TMA1ST flag
TEST DATA[17]:	receive timeslot inhibit
TEST DATA[31:18]:	'0'

5 Functional Test Specification (Macrolevel)

(use Checklist C06 "Function Checklist - Comparison of Specification vs. Circuit"
http://www.hl.siemens.de/lognet/hl_ta/dhb/f.htm)

Appendix M

Title:
Timeslot Assigner Transmit V2.2.1

1 Introduction

1.1 Overview

The Timeslot Assigner Transmit (macro name key: TT_) maps the incoming transmit timeslots (maximal 32 ports x 32 timeslots = 1024 timeslots and for the M256F 28 ports x 24 timeslots = 672 timeslots) to logical channels and provides the channel relevant information to the protocol machine interface. The TSAT provides the timeslot relevant information belonging to this channel to the transmit port interface.

1.2 Features

- coordinates requests of 32 ports [M256F: 28ports] by built-in arbitration
- arbitration priority from low (port n = 0) to high (port n=31)
- channel programming by indirect access through TFPI Slave Interface
- data output FIFO buffering for adjustment of further data processing
- programmable channel assignment per timeslot
- programmable mask per timeslot
- programmable timeslot inhibit flag per timeslot
- remote channelwise loop, one channel at a time (loop RD -> TSAR -> TSAT -> TD)
- remote portwise loop, one port at a time (loop RD --> TSAR --> TSAT --> TD)
- loop supported by a jitter attenuator, consisting of a 512 bit FIFO with slip function
- programmable "TMA1ST flag" (tmafirst flag) to identify the 'first' timeslot for TMA mode channels
- FPI target interface
- performance: SYSCLK up to 70 MHz
- number of gates:
- area:
- power consumption:
- scan path
- RAM (Timeslot Assigner Transmit Parameter Table: TATPT) built in selftest
- RAM (Timeslot Assigner Transmit Data Buffer: TATDB) built in selftest
- RAM (Timeslot Assigner Transmit Fifo: TATFIFO) built in selftest

1.3 System Integration

The TSAT has four interfaces:

- FPI Slave (TFPI) Bus
- parallel REQ/GNT interface to (T)XPI functions (accept incoming timeslot information, provide data to transmit port)
- REQ/GNT interface to PMT functions (provide channel information to processing functions)

- alternate input for remote loop (for one channel or port at a time)

The TSAT receives the port and time slot information from the (T)XPI block. The incoming timeslot is mapped to logic channel information and provided to the protocol machine. Finally the channel relevant data from the protocol machine and mask bit field information is provided to the transmit port interface. Additionally this data can be selected for one single channel or port from the remote loop interface. The mapping information is programmed to the TSAT via the FPI Slave interface.

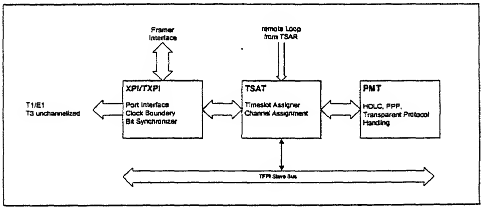


Figure 1:
System integration

1.4 Known Restrictions

1) necessary minimum SYSCLK to serve 32 ports at E1 data rate (32 timeslots), without regarding synchronization losses

one access request per timeslot; i.e. 2.048Mbit x 32 ports / 8bits/timeslot = 8 Mbyte/s and 4 clock cycles for processing (arbitration, mapping, data transfer) 33MHz SYSCLK would be sufficient.

in m256f application, i.e T1 1.544Mbit x 28ports / 8bits/timeslot = 5.4 Mbyte/s and 4 clock cycles for processing, the minimum frequency for SYSCLK = 22 MHz.

2) priority and waitstates for indirect read/write access from the FPI slave interface to the TSATPT RAM.

For operation with maximum data rate, data processing takes every forth clock cycle access to the TSATPT RAM. Else the TFPI slave interface has access to the

TATPT. Thus an indirect write access can be delayed 1 clock cycle for write and 3 clock cycles for read.

3) restrictions regarding the remote loop:

a) Channelwise loop: same port for receive and transmit direction, same number of timeslots activated (inhibit bit = 0), identical mask bit field for all timeslots that are activated activated

b) Portwise loop: same port for receive and transmit direction, only one channel is supported which comprises all timeslots, inhibit bits are not allowed, identical mask bit field for all timeslots

4) timeslot to channel assignment programmed in the parameter table for unchannelized mode .

for unchannelized mode all timeslots of a port have to be assigned to one channel,. valid timeslots are timeslot 0 to 23.

2 Functional Description

2.1 Block Diagram

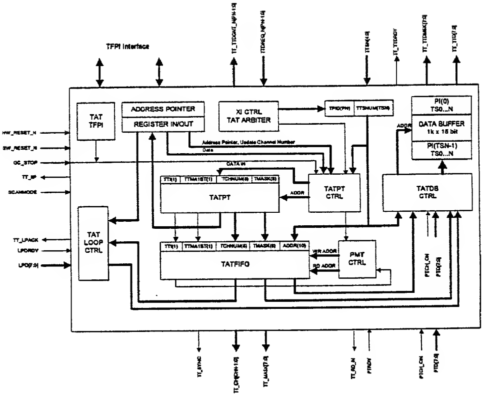


Figure 2.1:
TSAT Block Diagramm

2.2 Normal Operation Description / Reset

With an active reset (HW_RESET_N or SW_RESET_N) all registers are set to their benign state and the TSATFIFO is reset to empty (there is no difference between HW and SW reset). After the reset signal has become inactive, the TSATPT RAM (TSAT Parameter Table) starts its self initialization procedure, resetting the RAM values to TI = '1' (inhibit timeslot), TMA1ST = '0' (no TMA sync timeslot), CHNUM=00_H (channel number 0) and MASK = 00_H (all bits masked). An initialisation procedure is also done for the buffer RAM to transmit port interface, setting the RAM values of data to TTD = FF_H and mask bits to TDMSK = 00_H (all bits masked)

During this initialization period the TSAT is hold in stop mode. The TSAT is inaccessible from all interfaces and doesn't generate any request itself. The active self init period is indicated by output TT_IIP = '1', which becomes '0', when both RAMs (TATPT and buffer RAM) have been initialized.

For fast initialization of the TATPT RAM an input GC_STOP is still asserted (reset value) and keeps the TSAT in stop mode. In stop mode the control logic assigns the TATPT access exclusively to the TFPI interface and normal operation to this RAM is suspended. Thus the initial programming of the TATPT can be achieved very fast since it is not interleaved by normal operation accesses. A request from the (T)XPI will be processed as in normal operation mode, but setting the data to TT_TTD = FF_H and mask to TT_TTMSK = 00_H (all bits masked). The TATPT is programmed now by writing to the two TSAT indirect access registers. The TATPT address corresponds to the time slot number and port number.

In normal operation mode (GC_STOP='0') TFPI re-programming requests to TSATPT are handled by the control logic as ordinary requests for access. Therefore they can be delayed up to 1 clock cycle for write until the access is granted. For read cyles valid data is delayed 2 additional clock cycles because the access to the TATPT is pipelined. Before reprogramming a channel to timeslot mapping configuration the corresponding channel must be turned off, in order to avoid intermediate TATPT states for timeslots of that channel. The TATPT contains the following entries per time slot (address):

TATPT.TI (inhibit transmit time slot)

TATPT.TTMA1ST (flag to be used by PMT in TMA mode)

TATPT.TCHNUM[7:0] (channel number for that timeslot)

TATPT.TMASK[7:0] (mask bits for that timeslot)

After deasserting the GC_STOP input signal, the TSAT is in normal operation mode. The arbiter now accepts requests from (T)XPI signals TTDREQ_N and grants the time slot number bus TTSN to the selected initiator by asserting the corresponding grant signal TT_TTDGNT_N. After deasserting of the corresponding TTDREQ_N signal (marks valid information on the bus) TTSN is sampled for internal processing. When the TT_TTDRDY line is active TSAT will provide 8 bit data TT_TTD and 8 bit mask field TT_TTDMSK.

To achieve high data throughput, data processing from XPI to PMT is pipeline oriented. The requests for data processing from (T)XPI are provided from a data buffer RAM with one entry for every timeslot of each port. After processing the read transaction to PMT the RAM location for this timeslot is updated.

The following information is received from the (T)XPI:
requests for data processing from the (T)XPI : TTDREQ_N[PN-1:0]
timeslot number bus of grantet (T)XPI : TTSN[4:0]

The following information is delivered to the (T)XPI:
grant signal to select one of the (T)XPI: TT_TTDGNT_N[PN-1:0]
transmit timeslot data for granted (T)XPI: TT_TTD[7:0]
transmit mask bit field for granted (T)XPI: TT_TTMSK[7:0]
finished data transaction for one of the (T)XPI: TT_TTDRDY

The timeslot number, which is received on bus TTSN is combined with the port ID, that the arbiter generates from the corresponding request line TTDREQ_N. The concatenation of timeslot number and port id point to a location in the TATPT. The corresponding contents are read and written into the TATFIFO. A transfer to the PMT is started by TSAT by activating the request line TT_RD_N. In the same cycle the channel number and the mask lines are activated. For the first active timeslot of each logical channel in TMA mode, the TSAT activates the TT_SYNC line. This TMA1ST info flag is used in the protocol machine transmit for synchronisation purposes to indicate the first timeslot in a frame, which is assigned to a multi-timeslot TMA channel. It must also be set for a single timeslot TMA channel. If TI (timeslot inhibit) is set, no request to the PMT will be generated for this timeslot. Nevertheless the timeslot entry in the TATDB for the (T)XPI request is updated, setting the data to TTD = FF_H and mask to TTMSK = 00_H (all bits masked).

The data transaction is finished by PMT when the PTRDY line is active. Data and channel status information from PMT are sampled and the timeslot entry of the corresponding channel in TATDB is updated. If the channel status signal from the PMT indicate the off status, the entry in the TATDB for the mask is set to TTMSK = 00_H, and for TTD is set to FF_H.

The following information is delivered to the PMT:
the assigned channel number: TT_CH[7:0]

the programmed mask for that timeslot: TT_MASK[7:0]
the TMA1ST info flag: TT_SYNC

The following information is received from the PMT:
the transmit data from the PMT: PTD[7:0]
"channel on" status of the PMT: PTCH_ON

2.3 Remote Loop

A basic channelwise or portwise remote loop is available from the TSAR to the TSAT. The payload of one (and only one) logical channel or port is mirrored from the receive part to the transmit part of the looped port. In SCM mode, the framing bits, CRC and spare bits are not looped. They are provided by the M256F framer and the facility data link controller. The channel number or port ID is programmed in the configuration register 2 CONF2. The loop is activated by setting the corresponding loop command bit located in CONF2. Receive timeslots of that channel or port matching the programmed channel number or port ID are written to the TSAR loop FIFO (64 x 1 byte) which is implemented as a circularly organized memory controlled by a read pointer and a write pointer. The loop FIFO acts as a jitter attenuator which compensates the clock jitter between receive and transmit clock.

As soon as data is available on the remote loop interface, the LPDRDY signal is activated by the TSAR. The data transfer will be processed with transmit timeslots of the channel or port matching the programmed channel number or port ID. For the looped channel or port no request to the PMT will be generated. The data coming from TSAR and mask information read from the TATFIFO is written into TATDB at the RAM location for the timeslot requested by TXPI. The loop is turned off by resetting the loop command bit. The LPDRDY signal is then deactivated, and the loop FIFO will be reset to empty.

Only one loop command bit, either for channel or port loop, should be set at a time. Generally, the port number or channel number has to be programmed first before setting the portwise or channelwise loop command bit.

In case of a channel loop, the channel characteristics in receive and transmit direction must be identical. The numbers of active receive and transmit timeslots of the looped channel must be identical. The mask bit fields in the active timeslots must be identical.

In case of a port loop, only one channel is allowed which comprises all timeslots. Inhibit bits for the looped port are not allowed.

The data transfer for the looped channel or port to the PMR is not affected.

Slip Function

After activating the loop, 32 receive data bytes are written to the loop FIFO until the LPDRDY signal is asserted by TSAR. During this loop FIFO initialization phase, LPDRDY is still deactivated. The timeslot entry in the TATDB is updated with $TTD=FF_H$ and $TTMSK=00_H$ (all bits masked). When LPDRDY is asserted by TSAR, TSAT starts reading the transmit data from the loop FIFO. The initial distance between the FIFO read pointer (RP) and the write pointer (WP) is 32 bytes. Due to a RCLK/TCLK clock jitter the RP may move towards WP. In case the distance between RP and WP is equal plus/ minus 1 byte a slip of the read pointer will occur. Provided RP is faster than WP a positive slip occurs (the previously received 31 bytes are read out twice). In case RP is slower than WP a negative slip occurs (the next 31 received bytes are skipped). The slip condition is checked with each access to the loop FIFO.

3 Macro Interfaces and Signal Description

All signals are active high until otherwise specified. Active low signals are designated by "_N" appended to their names. To make the design as re-usable as possible, a bus signal whose width is application dependent is specified with one of the following parameters:

Parameter name	Bus Type	Typical Value (Bits) M256F
PN	max. port number bus	5
TSN	max. timeslot number bus	5
CHN	max. channel number bus	8
PAD	port address	5
DB	data bus width	32
AB	address bus witdh	30

3.1 Signal Description

3.1.1 Global Signals

Signal Name	Dir ecti on	Type	Tsu/ Thld Td	meaning/comment special characteristics
SYSCLK	i			internal system clock
HW_RESET_N SW_RESET_N	i i			general SW and HW Reset
SCANMODE	i			SCAN Test mode
GC_STOP	i			external init keeps TSAT in stop mode, exclusive access to TATPT RAM from TFPI
TT_IIP	o			TSAT initialization in progress following HW reset. TSAT is not availble when this signal is asserted. This signal will remain high for several clocks following release of TSAT reset(s).

3.1.2 Transmit Port Interface

The XPI(TXPI) interface consists of the following signals

Signal Name	Dir ecti on	Type	Tsu/ Thld Td	meaning/comment special characteristics
TTDREQ_N[PN-1:0]	i			*request from transmit port n=index for servicing/* reading 8 bit data and 8 bit mask field
TTSN[TSN-1:0]	i			time slot bus containing the time slot number for the data TT_TTD and mask *TT_TTDMSK (valid* time slot numbers are 0 to 23 for T1, 0 to 31 for E1 **and for unchannelized mode all timeslots of a port have to be assigned to one channel**
TT_TTDGNT_N[PN-1:0]	o			ttsn bus is granted to transmit port n=index
TT_TTDRDY	o			TSAT has finished data transfer in current clock cycle
TT_TTD[7:0] TT_TTDMSK[7:0]	o			data bus containing 8 bit time slot data and 8 bit mask bit field

Description of the (T)XPI interface protocol:

As soon as any port TXPI[n] needs 8 bit data for transmit, it asserts the request signal TTDREQ_N to the arbiter part of the TSAT, in order to request service. The arbiter then grants by activating the respective grant signal TT_TTDGNT_N one port access to the bus TTSN (transmit time slot number). TTDREQ_N is deasserted to indicate valid data in the next clock cycle on the bus TTSN. TT_TTDRDY is activated to indicate that the current data transfer has finished in the current clock cycle and valid data on the busses TT_TTD and TT_TTDMSK is available.

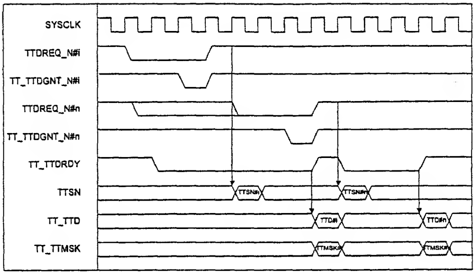


Figure 3.1.2:
Data Transfer from XPI (TXPI) to TSAT

3.1.3 Protocol Machine Transmit Interface

The PMR interface consists of the following signals:

Signal Name	**Dire ctio n**	**Type**	**Tsu/Thld Td**	**meaning/comment** **special characteristics**
TT_RD_N	o			request for servicing/ reading data
TT_CH[CHN-1:0]	o			logical channel number
TT_MASK[7:0]	o			8 bit mask field to enable single bits of TSA data
TT_SYNC	o			logical channel synchronization line in TMA mode
PTRDY	i			PMT finished data transaction
PTD[7:0]	i			8 bit data from read transaction
PTCH_ON	i			"channel on" status signal from PMT '0' -> channel is not active '1' -> channel is actice

Description of PMT interface protocol:

Whenever data in the TATFIFO is available a transfer is started by TSAT asserting the signal TT_RD_N. In the same cycle channel number and mask information is provided. For the first active timeslot of each logical channel in TMA mode, the TSAT also actives the TT_SYNC line. The information is hold stable until PMT finished data transaction by activating the line PTRDY with valid data on the bus PTD and channel active indication by signal PTCH_ON.

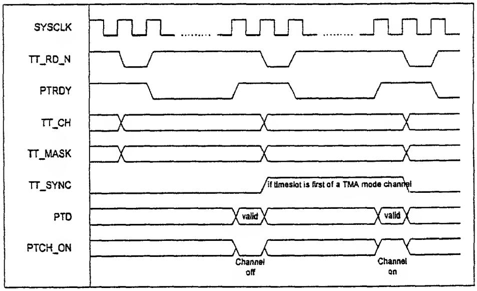


Figure 3.2:
Data transfer from TSAT to PMT

3.1.4 TSA Receive Interface (for remote loop)

TheTSA reveive interface consists of the following signals

Signal Name	Dir ecti on	Type	Tsu/ Thld Td	meaning/comment special characteristics
TT_LPACK	o			acknowledge from TSAT data has been read
LPD[7:0]	i			8 bit timeslot data of channel or port which is in remote loop operation
LPDRDY	i			indicates valid data and mask bit field

Description of TSA receive interface protocol:

The remote loop is activated by programming channel number or port ID and the corresponding loop bit CRLP or RLP set in the global mode register. As soon as the loop fifo in TSAR has filled up, LPDRDY signal is asserted. The loop data LPD is read and written by TT_LPACK signal. The remote loop is deactived by resetting the corresponding loop bit.

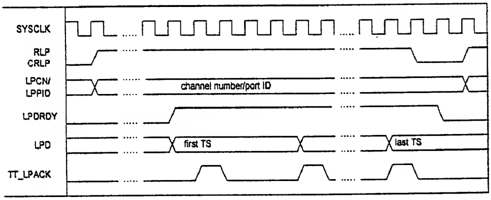


Figure 3.3:
Data transfer on the loop interface

3.1.5 FPI Slave Interface

In the following sections, "Flexible Peripheral Interconnect (FPI) Bus compliant" means that the specified bus uses a subset of the FPI features and satisfies the basic address and data cycle. Not all FPI signals are implemented because default values are sufficient for the application i.e. they can be coded as constants in the hardware. Refer to the FPI bus specification and FPI + Target Template Bus for details of the complete bus.

The TFPI interface consists of the following signals

Signal Name	Dir ecti on	Type	Tsu/ Thld Td	meaning/comment special characteristics
TFPI_RD_N	i			Read control input
TPFI_WR_N	i			Wnte control input
TFPI_A[AB-1:2]	i			Address input
TFPI_D[DB-1.0]	i			Data input
TFPI_RDY	i			Ready input
VG_TFPI_SEL_N	i			(Macro) select input (broadcast programming virtual global register)
TFPI_SEL_N	i			(Macro) select input (macro specific programming)
TT_TFPI_D[DB-1:0}	o			Data output
TT_TFPI_D_EN	o			Data enable (active high)
TT_TFPI_RDY	o			Ready output
TT_TFPI_RDY_EN	o			Ready enable

4 Register Description

4.1 Register Overview

Register Overview Table : TSAT V2.1

Register ID	Access	Absolute Address cs_n & a(7:2)	Reset Value	Comment
CONF2	R/W	44_H	00000000_H	(virtual global) configuration register 2
TSAIA	R/W	70_H	00000000_H	(macro specific) timeslot assignment indirect access register
TSAD	R/W	74_H	02000000_H	(macro specific) timeslot assignment data register
TAC	R/W	58_H	00000000_H	(virtual global) test command register
TD	R/W	$5C_H$	00000000_H	t(virtual global) test data register

4.2 Detailed Register Description

4.2.1 (Virtual Global) Configuration Register 2 (CONF2)

Access	: read/write
Address	: 00000044_H
Reset Value	: 00000000_H

31															16

15	14		12 8	7 0
CRLP	RLP		LPPID[4:0]	LPCN[7:0]

Note: *This register is common for TSAT and other macros. The macro TSAT just collect those data bits from the written dword (see above), that are relevant for macro TSAT. With a read access to this register, the macro TSAT drive only a value for the relevant bits. All other bits are set to '0'.*

LPCN[7:0]: Channel selection for channelwise remote loop

LPPID[4:0]: Port selection for portwise remote loop

CRLP: '1'--> channelwise remote loop on channel selected by LPCN[7:0]

RLP: '1' --> portwise remote loop on port selected by LPPID[4:0]

4.2.2 Timeslot Assignment Indirect Access Register (TSAIA)

Access	: read / write
Address	: 00000070_H
Reset Value	: 00000000_H

31								23							16
DIR	0	0	0	0	0	0	0	AI	0	0	0	0	0	0	0

15				7			0
0	0	0	PORT[4:0]	0	0	0	TSNUM[4:0]

Address:

This register specifies the base address for access to the TATPT (TSAT parameter table RAM). The address is composed of timeslot number and port id. The address bit field TSNUM[4:0] corresponds to the timeslot number of port which is selected by the address bit field PORT[4:0].

TSNUM[4:0]	timeslot number which is affected by access (T1 = 0..23, E1 = 0..31)
PORT[4:0]	port number which is affected by access (port 0..31)

DIR: transmit/receive direction

1--> Timeslot Assigner Transmit

0--> Timeslot Assigner Receive

AI: auto increment mode

If set when writing a start address to the address pointer, the TSNUM bit field of the address will be automatically incremented (and wrapped at the maximum value x1F) after each read or write access to the data register.This simplifies the transfer of data blocks because the address register for a certain port has to be written only once at the beginning of a data transfer.

Read access to TSAIA

On read access to TSAIA TSAT responds, when DIR is set to '1' (which needs a write access first in order to set DIR). When DIR is set to '0' the TSAR should respond. The macro, which is not accessed, should respond with all '0'.

In case having used the autoincrement mechanism before, the returned value should be the actual address, i.e. the autoincremented timeslot number.

4.2.3 Timeslot Assignment Data Register (TSAD)

Access	: read/write
Address	: 00000074_H
Reset Value	: 02000000_H

31						25	24								16
0	0	0	0	0	0	TTI	TTMA 1ST	0	0	0	0	0	0	0	0

15	0
TCHNUM[7:0]	TMASK[7:0]

The data register is a port into the TATPT RAM for read and write access. The RAM address is specified by the address bit field in the timeslot assignment indirect access register. The TATPT RAM is initialized following hw_reset_n.

TMASK[7:0] transmit mask bit field for bit rate adaption, mask bit = '1' bit is enabled, mask bit = '0' bit set to tristate (reset value = 00_H)

TCHNUM[7:0] transmit channel number, maps the corresponding timeslot to this logical channel (reset value = 00_H)

TTMA1ST transmit TMA first (for TMA synchronisation purpose), TMA = '1' indicates first timeslot in a logical TMA channel (reset value = '0')

TTI transmit timeslot inhibit, TTI = '1' no request to the protocol machine for this timeslot, timeslot is set to tristate (reset value = '1')

4.2.4 (Virtual Global) Test Command Register (TAC)

Access	: read / write
Address	: 00000058$_H$
Reset Value	: 00000000$_H$

31			25	24	23 16
MID	0	0	0	AI	COMMAND

15				0
0	0	0	0	ADDRESS

Refer to the Implementation Specification M256F for details to the TAC register.

MID:	Macro ID Code (**TSAT: "0111"**)
AI:	auto increment function
ADDRESS:	internal address
CMD:	command (select of RAM or register)

CMD:

Specifies the access to memory block or register being addressed. Defined command types are :

00000001 : TATPT testmode read (TSAT parameter table RAM)

10000001 : TATPT testmode write

00000010 : TATDB testmode read (TSAT timeslot data buffer RAM)

10000010 : TATDB testmode write (TSAT timeslot data buffer RAM)

On read access to TAC the macro which is selected via MID should respond to this access. All other macros should respond with all '0'.

4.2.5 Test Data Register (TD)

Access	: read/write
Address	: $0000005C_H$
Reset Value	: 00000000_H

31	16
TEST DATA	

15	0
TEST DATA	

Refer to the Implementation Specification M256F for details to the TD register.

The TD register is a port into theTSAT RAMs TATPT (TSAT parameter table RAM) and TATDB (TSAT data buffer RAM) for read and write access. The selet of RAM and access type is specified by the command bit field in the register TAC. The RAM address itself is specified by the address bit field in the register TAC.

TATPT: TSA transmit parameter table RAM

TEST DATA[7:0]:	transmit mask bit field
TEST DATA[15:8]:	transmit channel number
TEST DATA[16]:	transmit TMA1ST flag
TEST DATA[17]:	transmit timeslot inhibit
TEST DATA[31:18]:	'0'

TATDB: TSA transmit data buffer RAM

TEST DATA[7:0]:	transmit data to XPI (TT_TTD)
TEST DATA[15:8]	transmit mask to XP (TT_TTMSK)
TEST DATA[31:16]:	'0'

5 Functional Test Specification (Macrolevel)

(use Checklist C06 "Function Checklist - Comparison of Specification vs. Circuit" http://www.hl.siemens.de/lognet/hl_ta/dhb/f.htm)

Appendix N

Title: Receive/Transmit Port Interface V2.3.1

1 Introduction

1.1 Overview

The XPI block (macro name key: XI_) interfaces the serial data streams on e.g. PCM formatted IOs and maps them to/from chip internal processing. It therefore performs serial/parallel conversion and transfers data between the serial clock system and the on chip system clock.

The XPI contains the subfunctions RXPI for receive direction and TXPI for transmit direction.

1.2 Features

- serial interface operating with gapped/non gapped serial clocks
- operation with strobed serial clocks **(future extension)**
- standard channelized mode (SCM) : 28 T1 (1.544MHz) / E1 (2.048 MHz) lines with chip internal framing function
- alternate channelized mode (ACM) : 16 x T1 (1.544MHz) / E1(2.048MHz) lines without chip internal framing function (E1/T1 arbitrarily portwise programmable)
- unchannelized mode (UCM) and mixed modes (SCM/UCM or ACM/UCM), UCM mode available for 16 ports
- SCM test mode: processing of 9 bit test frames (in T1 mode only, ports #16-27 only)
- serial data sampling/transmitting with rising/falling edge of clocks (ACM, SCM and UCM)
- SCM: sampling the transmit sync puls with rising/falling edge of transmit clock
- Optional disable/enable of external transmit synchronization in SCM mode
- ACM: sampling the receive/transmit sync pulses with rising/falling edge of clocks
- ACM: programmable TD/RD bitshift of +3/-4 bits relative to the sync pulse
- serial data multiplexable between normal and internal loop input
- Looped timing: In SCM mode RCLK(n) can be chosen as clock source for the transmit part of port#n
- supervision of frame synchronization conditions
- Signaling of the On Chip Framer receive synchronization for one port out of 28(SCM) at the serial interface or, alternatively, signaling of the effective transmit clock of port#0
- PCM-, HDLC-, framer- and interrupt-interface
- performance: serial clock up to 20MHz (port 0: 60MHz) ; SYSCLK up to 70 MHz

  (in M256F application only the port 0 provides a data rate up to 60 MBit/s)
- number of gates:
- area:
- power consumption:

- full scan path for the sysclk domain (in M256F no scan path for serial clock domains)

1.3 System Integration

RXPI interfaces:

- n serial PCM interfaces [M256F: n=28]
- one parallel (8bit data + 5bit timeslot number) REQ/GNT interface to TSAR arbiter (provides receive data and corresponding timeslot information)
- n single bit interfaces [M256F: n=28] to the FRAMR (provides framer data)

TXPI interfaces:

- n serial PCM interfaces [M256F: n=28]
- one parallel (8bit data + 8bit mask + 5bit timeslot number) REQ/GNT interface to TSAT arbiter (provides timeslot information for selected port)
- n single bit interfaces [M256F: n=28] to the FRAMT (provides framer data and transmit sync pulse)

Shared interfaces:

- one FPI Slave (TFPI) Bus
- one Interrupt Bus

The RXPI subblock is used at the chip interface, in order to prepare the serial data for further processing. The PCM data is sampled with its respective clock and synchronized to the system clock domain. The frame alignment and DL support is provided by the framer receive part (FRAMR interface). The data provided with corresponding port and timeslot information is transferred to the timeslot assigner (TSAR interface).

The TXPI subblock is used to make serial transmit data available at the PCM interface. The data which are received from the TSAT macro, 8 bit in parallel, is composed to the desired frame and hyperframe structure. The DL and framing information is provided by the framer transmit part (FRAMT interface). The transmit data is transferred from the system clock domain to the PCM transmit clock domains of the corresponding ports.

Configuration of the RXPI and TXPI is done via slave accesses to the TFPI interface. Interrupt conditions are flagged by writing interrupt vectors to the interrupt interface.

One RXPI#n and one TXPI#n subblock corresponds to one serial interface. Up to 32 instances of the RXPI#n and TXPI#n can be combined to form the block RXPI and TXPI, respectively.

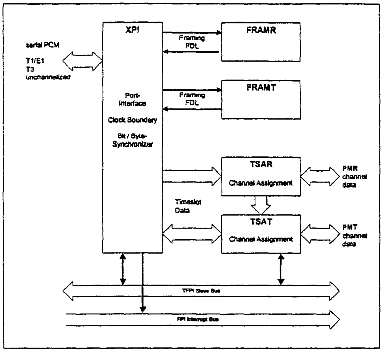


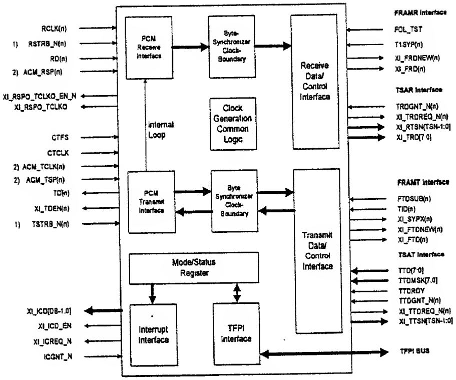
Figure 1.3: System Integration

1.4 Known Restrictions

1) For synchronization purposes, the frequency ratio SYSCLK / RCLK(TCLK) must be at least 3.5 : 1 for the ports #1 - 15, i.e. the max. RCLK(TCLK) frequency in UCM is approximately 9 MHz, provided that SYSCLK frequency = 33 MHz. Maximum serial frequency of the ports#16-27 is 2.048 MHz, regardless of the operation mode.
2) For unchannelized mode (UCM) all timeslots of a port have to be assigned to one channel. In UCM the XPI macro generates the T1 time slot scheme, i.e. 24 time slots (round robin) are given to the TSA.

2 Functional Description

2.1 XPI Block Diagram



Note 1): for M256F application fixed to "0"
Note 2): for M256F application ACM mode for port#n = 0...15 only

Figure 2.1: XPI Block Diagram

2.2 Normal Operation Description / Reset RXPI and TXPI

After HW or SW reset all registers are set to the benign state. There is no difference between HW and SW reset concerning the effect on the XPI hardware. The interface mode will be set to: 28xT1 (SCM) as defined by the reset value of the global configuration register **CONF1.GPM** .

Receive Direction:

All receive lines are disabled by the reset status of the receive enable register **XPI.REN[31:0]** = 00000000_H. Each bit of register REN corresponds to the respective receive port. No requests to TSAR and FRAMR are generated in this state, however the corresponding data bits at these interfaces have defined values ('0').

Start of operation is achieved by programming a REN bit to '1'. In SCM mode the requests to the FRAMR will start, but requests to TSAR will not start until the FRAMR indicates the synchronous state to (R)XP.

If the general port mode has been programmed to ACM (16 T1/E1 ports) by the register **CONF1.GPM** (simultaneously for the complete XPI !), the ports 0 to 15 expect the sync signal not to come from FRAMR, but from the macro input ACM_RSP(0...15). In this mode REN enables/disables only requests to TSAR. Requests to the FRAMR are generally disabled in ACM mode. Synchronization is achieved when two conditions are valid:

a) enabling of a port by setting the corresponding REN bit to ´1´;

b) at least one external receive sync pulse on ACM_RSP(n) has arrived to trigger the XPI timeslot counter. The timeslot counter is sensitive to RSP immediately after the reset lines are switched inactive, independent of an active or inactive REN.

After the synchronous state is achieved, the (R)XPI generates requests to the TSAR. If TSAR is still in stop mode or set to inhibit (timeslot), the requests are granted but discarded inside TSAR.

Transmit Direction:

In transmit direction the transmit enable register **XPI.TEN[31:0]** is also reset to 00000000_H, thus tristating the corresponding output ports. Each bit of the register TEN corresponds to the respective transmit port. The default interface mode is 28xT1 as defined by the reset value of GPM.

No requests to TSAT and FRAMT are generated in this state, however the corresponding data bits at these interfaces have defined values ('0'). When a bit TEN for a port is set to '1' and additionally, the transmit timeslot counter is triggered by at least one external trigger event ,(T)XPI has achieved the synchronous state.

With the next external trigger event, the (T)XPI will start requests to FRAMT and TSAT. If TSAT is still in stop mode or set to inhibit (timeslot), the requests are granted and answered to (T)XI with mask bit field set to '0' (all bits disabled).

The TXPI sends the data received from TSAT on the PCM line for this port. In SCM mode, the transmit timeslot counter is sensitive on CTFS which synchronizes all of the 28 ports (exception: looped timing and CTFS disable, s. section "Looped Timing" below).

If the general port mode has been programmed to ACM (16 xT1/E1) (ACM) via GPM (simultaneously for the complete XPI !), the ports 0 to 15 expect the sync signal not to come from CTFS input pin, but from the macro input ACM_TSP(0...15). In this mode TEN enables/disables only requests to TSAT. Requests to the FRAMT are generally disabled in ACM mode. The transmit clocks are provided via ACM_TCLK(0...15).

Receive/Transmit Direction:

All ports , which are activated (28 in SCM or 16 in ACM), can be selected separately to operate in unchannelized mode, by appropriate programming the port mode register (XPI.PMR). Physically, there exists one port mode register for each port.
In unchannelized mode no requests to the framer are generated from the respective ports.The unchannelized mode performs transparent data transfer without frame alignment. The framer function and any synchronization supervision is disabled in this mode. In UCM the XI macro generates the T1 time slot scheme, i.e. 24 time slots (round robin) are given to the TSA.

All interrupts generated by the ports (sync/async interrupts) are masked after reset, since the reset value of port mode register bit TEIM and REIM (transmit/receive sync error interrupt mask) is set to '1' (s. section "Port Interrupts").

After reset CTCLK is selected as global transmit clock for all output ports due to the reset value of CONF1.GPM = 0 (SCM mode, s. Figure 2.2.a). After switching to ACM mode (GPM = '1') each transmit clock domain #n (n = 0..15) is driven by the corresponding transmit clock ACM_TCLK(n). In the non loop case, each receive clock domain #n is clocked by the corresponding receive clock rclk(n) (n = 0..28 for SCM, n = 0..15 for ACM).

An internal test loop (TD out-> RD in XPI) is activated by programming the register CONF2 setting ILP to '1'. The loop function is available for only one port at a time, the port number being defined by CONF2.LPPID[4:0]. The respective transmit data and the transmit strobe information is sent to the corresponding receive port. In SCM mode, the receive part of the loop port is clocked by CTCLK. The on chip framer provides the receive frame synchronization. In ACM mode, the receive clock for the loop port #n (n =

0...15) is generated from the respective transmit clock, ACM_TCLK(n) (s. clock multiplexer, Figure 2.2.b). The receive frame synchronization corresponds to the transmit synchronization which is triggered by ACM_TSP(n).

For test purposes, the receive synchronization info provided by the on chip framer is visible on XI_RSPO_TCLKO for one port at a time. This test function is enabled by setting CONF2.RSPEN to '1'. The sync pulse is visible with a latency of 1 frame. The port selection is done via CONF2.SPAD(4:0). If CONF2.RSPEN is reset to '0' the effective transmit clock of port #0 is visible on XI_RSPO_TCLKO.

PCM Tristate Handling (Transmit Direction)

SCM mode, T1/E1: During and after an active sw or hw reset the PCM transmit data lines are tristated. When the transmit enable register (TEN) is enabled and the synchronous TXPI state is achieved, the first non tristate bit on the PCM data line is the first bit requested from TSAT. Mask information coming from TSAT is ignored, i.e. even data bits accompanied by active mask bits are driven actively on the PCM line. One or two transmit clock cycles after TEN is disabled, the PCM data line is switched to tristate.

UCM mode in case CONF1:GPM = 0: Apart from the fact that a synchronization phase does not exist, the tristate handling is the same as in T1/E1 mode.

ACM mode, T1/E1: The tristate behaviour is identical to the behaviour in SCM mode with the exception that data bits masked by TSAT are tristated.

UCM mode in case CONF1:GPM = 1: The tristate behaviour is identical to the behaviour in UCM mode (CONF1:GPM = 0) with the exception that data bits masked by TSAT are tristated. However, in UCM mode TSAT mask bits should be generally avoided.

Looped Timing

In SCM mode, the clock source for the transmit part of port#n (n = 0, ... , 27) can be switched from CTCLK to the corresponding receive clock RCLK(n), setting PMR.LT of port#n to '1'. An arbitrary subset of the transmit ports can be switched to looped timing. In case of an activated looped timing, the external receive frame synchronization via CTFS is disabled, i.e. the transmit frame counters are no longer triggered by CTFS pulses; they are free running. The supervision of the transmit synchronization is suspended (s. section "Port Interrupts").

The suspension of the external transmit frame synchronization incl.sync-async interrupt generation can also be achieved by setting PMR:CTFSD to '1'. However, in case of

PMR.CTFSD = '1' and PMR.LT = '0', the serial transmit part is operating with the regular SCM transmit clock CTCLK.

FDL Test Mode

For FDL test reasons a short frame test mode is implemented to yield higher F-bit data rates in order to reduce FDL verification effort. This test mode can be activated only for the ports #16 to #27 (SCM only ports), provided that

a) the dedicated XPI input pin FDL_TST is set to '1',

b) CONF1.GPM = '0' (SCM mode)

c) PMR.PCM = "0000" (T1 mode) for the enabled ports.

In short frame mode, the frame length is 9 bit, bit #0 treated as the framing bit. Apart from the frame length, the complete frame processing is the identical to the regular T1 SCM frame processing.

Port Interrupts

Port interrupts indicate the synchronous or asynchronous state of a port. Immediately after enabling the port interrupts by resetting the interrupt mask bits, port interrupts are generated indicating the current sync or async states. After this initial interrupt generation, a further interrupt occurs only when the state of a port changes from sync to async or vice versa. The reset value of the receive and transmit port state is async.

State Transitions

a) Standard Mode

A transmit port changes to the synchronous state, if common transmit frame synchronization is enabled and the number of bits between two synchronization pulses is equal to the number of frame bits of the selected mode or is equal to a multiple of that number. Immediately after reset, already the first CTFS pulse causes the transmitter to change to the synchronous state.

In case the common transmit frame synchronization is disabled, i.e. the looped timing bit or the CTFS disable bit of a port is set in PMR, the initial asynchronous state will not be left.

A transmit port changes to the asynchronous mode if the number of bits between two synchronization pulses is not equal to a multiple of the number of frame bits of the selected mode.

A receive port changes to the synchronous state, if the number of bits between two synchronization pulses generated by the port related framer is exactly equal to the number of frame bits of the selected mode. Immediately after reset, already the first framer pulse causes the receive port to change to the synchronous state.

A receive port changes to the asynchronous state if the number of bits between two framer synchronization pulses is not equal to the number of frame bits of the selected mode. The receive port expects an on-chip framer sync pulse for each frame.

b) Alternate Mode

A port changes to the synchronous state if the number of bits between two synchronization pulses is equal to a multiple of the number of frame bits of the selected mode. Immediately after reset, already the first synchronization pulse causes the port to change to the synchronous state. This holds for both directions, receive and transmit.

A port changes to the asynchronous state if the number of bits between two synchronization pulses is not equal to a multiple of the number of frame bits of the selected mode (receive and transmit direction).

In general, a state transition occurs only in T1 and in E1 mode. In unchannelized mode no supervision of frame synchronization is performed. The initial asynchronous state will not be left.

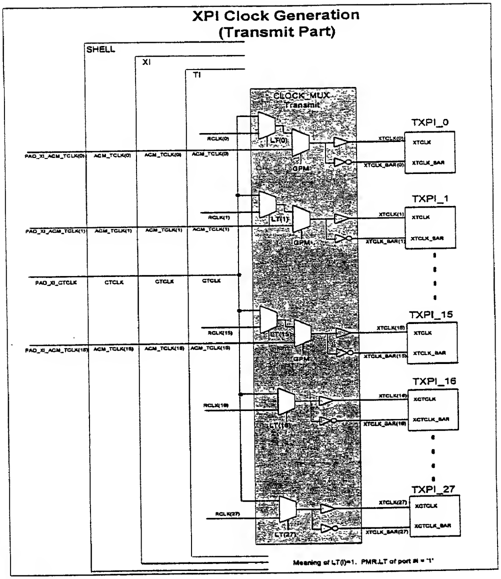


Figure 2.2a:

XPI Transmit Clock Generation

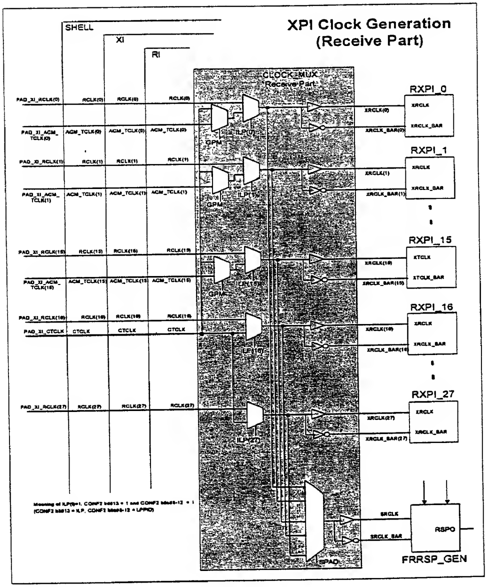


Figure 2.2b:

XPI Receive Clock Generation

3 Macro Interfaces and Signal Description

All signals are active high until otherwise specified. Active low signals are designated by "_N" appended to their names. To make the design as re-usable as possible, a bus signal whose width is application dependent is specified with one of the following parameters:

Parameter name	Bus Type	Typical Value (Bits) M256F
PN	max. port number	28
TSN	max. timeslot number bus	5
PAD	port address	5
ICD	max. interrupt vector	10
DB	data bus width	32
AB	address bus witdth	32

3.1 (Bus-) Interfaces and Protocols

3.1.1 Global Signals

Signal Name	Dire ctio n	Type	Tsu/Thld Td	meaning/comment special characteristics
SYSCLK	i			System Clock
HW_RESET_N	i			Hardware Reset
SW_RESET_N	i			Software Reset
SCANMODE	i			Scan Test Mode
M256_MODE	i			Switch to hardwired ACM mode

3.1.2 Timeslot Assigner Receive Interface

The TSAR interface consists of the following signals:

Signal Name	Direction	Type	Tsu/Thld Td	meaning/comment special characteristics
TRDGNT_N[PN-1:0]	i			data/tsnum bus is granted to port n=index
XI_TRDREQ_N [PN-1:0]	o			request from port n=index for servicing/reading 8 bit data
XI_TRD[7.0]	o			data bus containing 8 bit serial/parallel converted time slot data (bit 0 is the first serial bit received, bit 7 is the last serial bit received)
XI_RTSN[TSN-1:0]	o			time slot bus containing the time slot number of the corresponding data XI_TRD (valid time slot numbers are 0 to 23 for T1, 0 to 31 for E1 . **For unchannelized mode all timeslots of a port have to be assigned to one channel. In UCM the XPI macro generates the T1 time slot scheme, i.e. 24 time slots (round robin) are given to the TSA.**

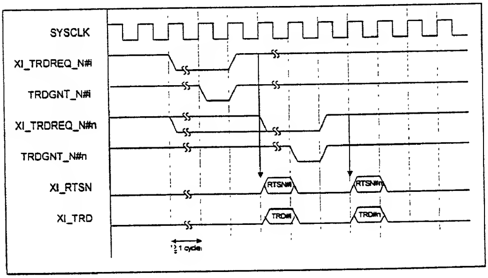


Figure 3.1.2:

TSAR Interface Timing Diagram

Description of TSAR interface protocol:

As soon as any port RXPI[n] has 8 bit data available for processing, it asserts the signal XI_TRDREQ_N to the arbiter part of the TSAR, in order to request service. The arbiter then grants TRDGNT_N the respective port access to the busses XI_TRD (data) and XI_RTSN (time slot number). TRDREQ_N is deasserted to indicate valid data on the busses XI_TRD and XI_RTSN. TSAR reads the bus information and deasserts TRDGNT_N.

The XI_RTSN and XI_TRD busses are generated by multiplexers which select the data of port#n according to TRDGNT_N#n. As a consequence, combinatorial delays are generated that may not be allowed application specifically (e.g. M256F).Therefore, the timeslot and data busses have to be registered within the macro (one additionally waitstait to TSAR).

3.1.3 Framer Receive Interface

The FRAMR receive interface consists of the following signals:

Signal Name	Direction	Type	Tsu/Thld Td	meaning/comment special characteristics
XI_FRDNEW[PN-1.0]	o	p * 1.		new receive data bit is valid (one signal per port)
XI_FRD[PN-1:0]	o			framer receive data (one bit per port)
T1SYP[PN-1:0]	i			framer receive control signal, to be evaluated when XI_FRDNEW is asserted, timeslot count and supervision of frame synchronization condition is triggered (s. Table 3.1.3)
FDL_RX_TST	i			If set to '1', the short frame test mode is active in SCM T1 mode

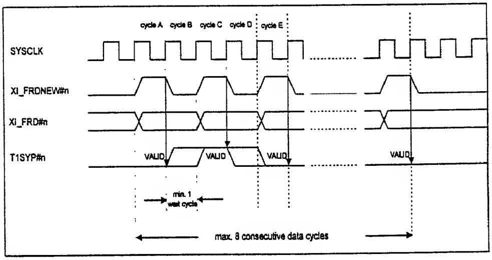


Fig. 3.1.3:
FRAMR interface timing diagram

Description of FRAMR interface protocol:

Each of the 28 PCM ports has its dedicated set of framer control and data signals. Fig.3.1.3 shows the signalling of one of the 28 port specific protocols which are driven completely independent.

Cycle A

A one cycle pulse on new bit indication signal (bit #n of XI_FRDNEW) is activated to request frame processing for the serial data bit of port #n, available on XI_FRD#n. The framer status signal T1SYP#n is sampled in response to the new bit indication. T1SYP#n must always be valid for each pulse of XI_FRDNEW#n.

Cycle B

A minimum of one wait cycle is performed before the next activation of XI_FRDNEW. At this time T1SYP#n is don't care for the XPI macro.

Cycle C

The next pulse of XI_FRDNEW is activated to request frame processing for the serial data bit of port #n, available on XI_FRD#n. T1SYP#n is sampled in response to the new bit indication. Table 3.1.3 shows the relevant signal combinations:

XI_FRDNEW	T1SYP	
1	0	T1 and E1: the current bit is to be passed to TSAR
1	1	T1: data to be deleted (frame bit# 0). The timeslot count and supervision of frame synchronization condition is triggered . E1: the current bit is to be passed to TSAR + sync check
0	x	no action

Table 3.1.3

Cycle D

A minimum of one wait cycle is performed before the next activation of XI_FRDNEW#n. At this time T1SYP#n is don't care for the XPI macro.

Cycle E

The next pulse of XI_FRDNEW#n is activated to request frame processing for the serial data bit of port #n, available on XI_FRD#n. At this time T1SYP#n must be updated.

Because of bytewise data processing, a maximum of 8 consecutive data cycles are possible in the worst case. However, the bytewise data processing is performed only for port#0 which has to meet high speed conditions. The ports no. 1 - 27 request frame processing only once in serial clock cycle (M256F application).

3.1.4 Timeslot Assigner Transmit Interface

The TSAT interface consists of the following signals:

Signal Name	Direction	Type	Tsu/Thld Td	meaning/comment special characteristics
XI_TTDREQ_N[PN-1:0]	o			request from transmit port n=index for servicing/ reading 8 bit data and 8 bit mask field
XI_TTSN[TSN-1:0]	o			time slot bus containing the time slot number for the data TTD and mask TTDMSK (valid time slot numbers are 0 to 23 for T1, 0 to 31 for E1 **For unchannelized mode all timeslots of a port have to be assigned to one channel. In UCM the XPI macro generates the T1 time slot scheme, i.e. 24 time slots (round robin) are given to the TSA.**
TTDGNT_N[PN-1:0]	i			XI_TTSN bus is granted to transmit port n=index
TTDRDY	i			TSAT has finished data transfer in current clock cycle
TTD[7:0] TTDMSK[7:0]	i			data bus containing 8 bit time slot data and 8 bit mask bit field

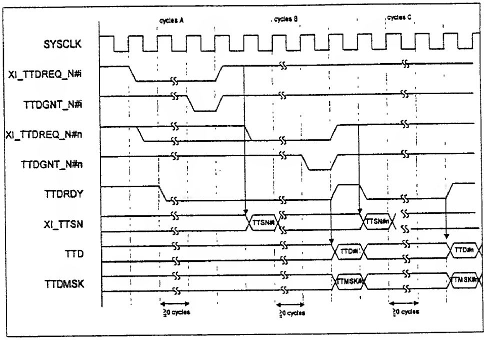


Figure 3.1.4:

TSAT Interface Timing Diagram

Description of the (T)XPI interface protocol:

As soon as any port TXPI[n] needs 8 bit data for transmit, it asserts the request signal XI_TTDREQ_N to the arbiter part of the TSAT, in order to request service. The arbiter then grants TTDGNT_N the respective port access to the bus XI_TTSN (transmit time slot number). XI_TTDREQ_N is deasserted to indicate valid data on the bus XI_TTSN. TTDRDY is asserted to indicate that the current data transfer has finished and valid data on the busses TTD and TTDMSK are available.

Each of the cycles A, B and C represents one or more clock cycles or it may be removed completely from the timing diagram.

The XI_TTSN bus is generated by a multiplexer which selects the data of port#n according to TTDGNT_N#n. As a consequence, combinatorial delays are generated that may not be allowed application specifically (e.g. M256F). Therefore, the timeslot bus has to be registered within the macro (one additionally waitstate to TSAT).

3.1.5 Framer Transmit Interface

The FRAMT transmit interface consists of the following signals:

Signal Name	Dire ctio n	Type	Tsu/Thld Td	meaning/comment special characteristics
XI_SYPX[PN-1:0]	o	p * 1		synchronous pulse transmit, to be evaluated when XI_FTDNEW is asserted (synchronized input CTFS)
XI_FTDNEW[PN-1:0]	o	p * 1		new transmit data bit is valid (one signal per port)
XI_FTD[PN-1:0]	o			framer transmit data, (one bit per port)
FTDSUB[PN-1:0]	i			framer transmit control signal, to be evaluated when XI_FTDNEW is asserted (s. Table 3.1.6)
TID[PN-1:0]	i			framer data to be inserted
FDL_TX_TST				If set to '1', the short frame test mode is active in SCM T1 mode

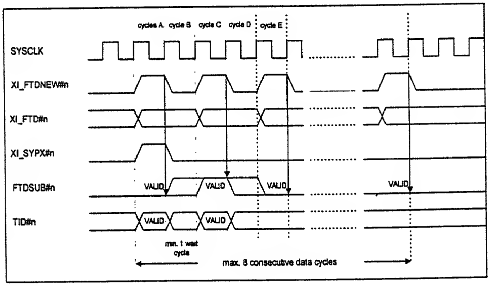


Fig.3.1.5:
FRAMT Interface Timing Diagram

Description of FRAMT interface protocol:

Each of the 28 PCM ports has its dedicated set of framer control and data signals. Fig.3.1.5 shows the signalling of one of the 28 port specific protocols which are driven completely independent.

The new bit indication line (bit #n of XI_FTDNEW) is activated to request frame processing for the serial data bit of port #n, available on XI_FTD#n.

Cycle A

A one cycle pulse on the new bit indication signal XI_FTDNEW#n is activated to request frame processing for the serial data bit of port #n, available on XI_FTD#n. FTDSUB#n coming from the framer is sampled in response to the new bit indication. FTDSUB#n must always be valid for each pulse of XI_FTDNEW#n.

An additionally activated XI_SYPX#n triggeres the FRAMT timeslot counter to indicate the beginning of a new frame. XI_SYPX marks bit #0 (F-bit or DL-bit) of the current frame which consists of 193 bits in T1 mode, resp. 256 bits in E1 mode. Table 3.1.5 shows the relevant signal combinations.

Cycle B

A minimum of one wait cycle is performed before the next activation of XI_FTDNEW. At this time FTDSUB#n and TID#n is don't care for the XPI macro.

Cycle C

The next pulse of XI_FRDNEW#n is activated to request frame processing for the serial data bit of port #n, available on XI_FTD#n.

The status signal FTDSUB is sampled in response to the new bit indication. Table 3.1.5 shows the relevant signal combinations:

XI_FTDNEW	XI_SYPX	FTDSUB	
1	0	0	data ready for PCM bit stream
1	0	1	required data has to be substituted by TID#n (e.g. test data coming from the framer)
1	1	0	sync bit location (F-bit), The framing bit or DL bit or CRC bit has to be provided on TID#n. The current data bit XI_FTD#n is postponed. Only used in T1 mode

XI_FTDNEW	XI_SYPX	FTDSUB	
1	1	1	sync check; data has to be substituted by TID#n. Only used in E1 mode bit #0.
0	x	X	no action

Table 3.1.5

<u>Cycle D</u>

A minimum of one wait cycle is performed before the next activation of XI_FTDNEW#n.

<u>Cycle E</u>

The next pulse of XI_FTDNEW#n is activated to request frame processing for the serial data bit of port #n, available on XI_FTD#n. At this time the status signal FTDSUB coming from the framer must be updated.

Because of bytewise data processing, a maximum of 8 consecutive data cycles are possible in the worst case. However, the bytewise data processing is performed only for port#0 which has to meet high speed conditions. The ports no. 1 - 27 request frame processing only once in serial clock cycle (M256F application).

3.1.6 Interrupt Interface

The interrupt controller interface consists of the following signals:

Signal Name	Dire ctio n	Type	Tsu/Thld Td	meaning/comment special characteristics
ICGNT_N	i			XPI is granted to interrupt bus
XI_ICREQ_N	o			request from XPI for interrupt servicing
XI_ICD[ICD-1:0]	o			interrupt bus (vector)
XI_ICD_EN	o			interrupt bus data enable

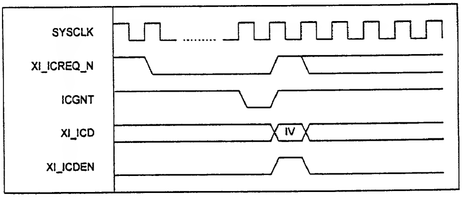


Figure 3.1.6:
Interrupt Controller Interface

Description of interrupt interface protocol:

XI activates XI_ICREQ_N to indicate an interrupt vector on XI_ICD. Data are valid in the next clock after ICGNT_N active. The XI could generate an asyc/sync interrupt at any PCM frame. For M256F application max. 56 interrupts (28 tx, 28 rx) are possible at a time.

Receive/Transmit Interrupt Vector

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
1	1	0	TDIR	0	Q2	Q1	Q0	0	0	0	0	0	0	SYN	ASYN

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	0	0	PORT[4:0]				

TDIR:
direction ('1': transmit; '0': receive)

Q(2:0):
queue number for interrupt vector (refer to register **CONF2** bit PORTIQ[2:0])

SYN
Synchronous state indication
SYN interrupts are generated
1) immediately after enabling the interrupts by resetting the interrupt mask bits when the port is in synchronous state (initial interrupt) or
2) in case the state changes from async to sync.

ASYN
Asynchronous state indication
ASYN interrupts are generated
1) immediately after enabling the interrupts by resetting the interrupt mask bits when the port is in asynchronous state (initial interrupt) or
2) in case the state changes from sync to async.

PORT(4:0):
port number for which the interrupt vector (IV) is written

3.1.7 FPI Target Interface

In the following sections, "Flexible Peripheral Interconnect (FPI) Bus compliant" means that the specified bus uses a subset of the FPI features and satisfies the basic address and data cycle. Not all FPI signals are implemented because default values are sufficient for the application i.e. they can be coded as constants in the hardware. Refer to the FPI bus specification and FPI + Target Template Bus for details of the complete bus.

The FPI Target interface consists of the following signals:

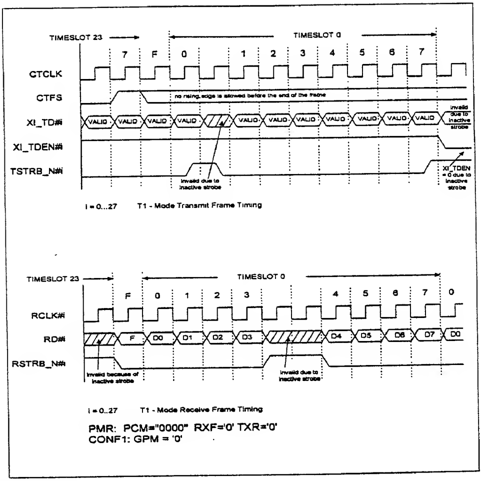
Signal Name	Direction	Type	Tsu/Thld Td	meaning/comment special characteristics
TFPI_RD_N	i			Read control
TFPI_WR_N	i			Write control
TFPI_A[AB-1:2]	i			Address bus
TFPI_D[DB-1:0]	i			Data bus, valid during write transfer
TFPI_RDY	i			End of transfer indicator
VG_TFPI_SEL_N	i			(Macro) select input (broadcast programming virtual global register)
TFPI_SEL_N	i			(Macro) select input (macro specific programming)
XI_TFPI_D[DBB-1:0]	o			Data bus, active during read transfer
XI_TFPI_D_EN	o			Data Enable
XI_TFPI_RDY	o			Ready Output
XI_TFPI_RDY_EN	o			Ready Enable

3.1.8 PCM Interface

The PCM interface consists of the following signals:

Signal Name	Direction	Type	Tsu/Thld Td	meaning/comment special characteristics
XI_RSPO_TCLKO	o			carries the receive synchronization info provided by the on chip framer for a user selectable port, if CONF2.RSPEN = 1. Each framer sync pulse is visible on XI_RSPO_TCLKO with a delay of 1 frame. If CONF2.RSPEN = 0, the effective transmit clock of port #0 is visible on XI_RSPO_TCLKO
CTFS	i			Common Transmit Frame Sync Signal This signal is used to synchronize the transmit lines that are clocked with CTCLK in SCM mode
RSTRB_N[27:0]	i			Strobe indicating valid receive data, the strobe signal is valid along the corresponding receive data RD
TSTRB_N[27:0]	i			Strobe indicating valid transmit data. The strobe signal is expected to be provided half a clock cycle before the corresponding transmit data XI_TD is generated.
CTCLK	i			Common Transmit Clock Reference Input
XI_TD[27:0]	o			Transmit Data DS1 Port 0..27
ACM_TCLK[15:0]	i			ACM Transmit clock for ports 0-15
RCLK[27:0]	i			Receive Clock for ports 0-27
RD[27-0]	i			Receive Data for ports 0-27
ACM_TSP[15:0]	i			ACM Transmit Frame Sync Signal for ports 0-15
ACM_RSP[15:0]	i			ACM Receive Frame Sync Signal forpPorts 0-15
XI_RSPO_EN	o			Pad Enable for XI_RSPO
XI_TDEN[27:0]	o			Tristate Control for Transmit Data

Refer to the Implementation Specification M256F for details of the complete interface. Operation with strobed serial clocks is for future extension and not relevant for the M256F application.


TIMESLOT 23
TIMESLOT 0
7
F
0
1
2
3
4
5
6
7
CTCLK
CTFS
no rising edge is allowed before the end of the frame
XI_TD#i
VALID
invalid due to inactive strobe
XI_TDEN#i
TSTRB_N#i
invalid due to inactive strobe
XI_TDEN = 0 due to inactive strobe
TIMESLOT 23
TIMESLOT 0
F
0
1
2
3
4
5
6
7
0
RCLK#i
RD#i
F
D0
D1
D2
D3
D4
D5
D6
D7
D0
RSTRB_N#i
invalid because of inactive strobe
invalid due to inactive strobe

i = 0...27 T1 - Mode Transmit Frame Timing

i = 0..27 T1 - Mode Receive Frame Timing

PMR: PCM="0000" RXF='0' TXR='0'
CONF1: GPM = '0'

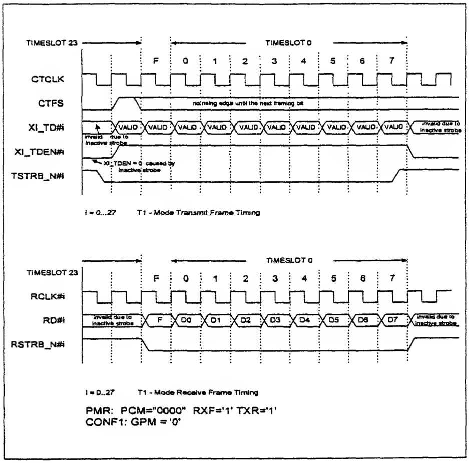
Figure 3.1.8a:

PCM interface timing for SCM: T1 with internal framing function

Port mode register (PMR): RXF = 0, TXR = 0 (LT = CTFSD = 0)

Transmit: generation of transmit data at falling CTCLK edge, sampling of the transmit sync pulse CTFS at rising edge, sampling of the transmit strobe TSTRB_N at falling CTCLK edge (TSTRB_N is provided half a clock cycle before the corresponding data is generated).

Receive: sampling of the receive data and receive strobe at rising RCLK edge.


TIMESLOT 23
TIMESLOT 0
F
0
1
2
3
4
5
6
7
CTCLK
CTFS
no rising edge until the next framing bit
XI_TD#i
VALID
invalid due to inactive strobe
XI_TDEN#i
XI_TDEN = 0 caused by inactive strobe
TSTRB_N#i
TIMESLOT 0
TIMESLOT 23
RCLK#i
RD#i
invalid due to inactive strobe
D0
D1
D2
D3
D4
D5
D6
D7
RSTRB_N#i

i = 0...27 T1 - Mode Transmit Frame Timing

i = 0..27 T1 - Mode Receive Frame Timing

PMR: PCM="0000" RXF='1' TXR='1'
CONF1: GPM = '0'

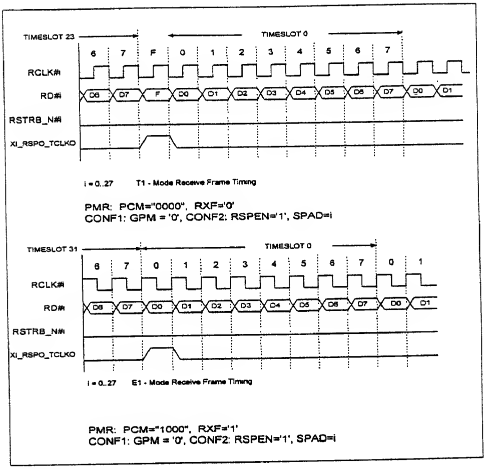
Figure 3.1.8b:

PCM interface timing for SCM: T1 with internal framing function

Port mode register (PMR): RXF = 1, TXR = 1 (LT = CTFSD = 0)

Transmit: generation of transmit data at rising CTCLK edge, sampling of the transmit sync pulse CTFS at falling edge, sampling of the transmit strobe TSTRB_N at rising CTCLK edge (TSTRB_N is expected to be provided half a clock cycle before the corresponding data is generated).

Receive: sampling of the receive data and receive strobe at falling RCLK edge.



i = 0..27 T1 - Mode Receive Frame Timing

PMR: PCM="0000", RXF='0'
CONF1: GPM = '0', CONF2: RSPEN='1', SPAD=i

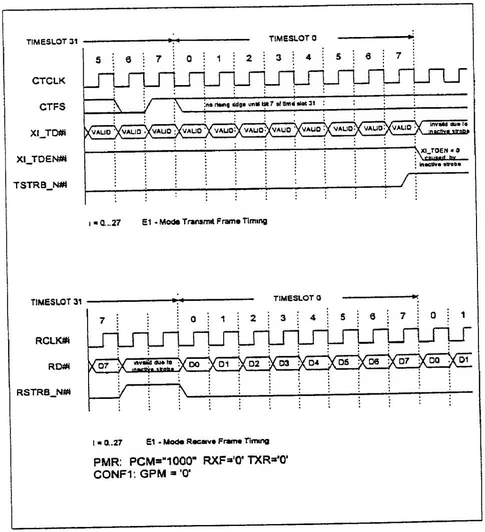
i = 0..27 E1 - Mode Receive Frame Timing

PMR: PCM="1000", RXF='1'
CONF1: GPM = '0', CONF2: RSPEN='1', SPAD=i

Figure 3.1.8c:

PCM interface timing for SCM: T1, E1 with internal framing function

XI_RSPO_TCLKO carries the receive synchronization info provided by the on chip framer for a user selectable port. The sync pulses are visible after the framer has reached its synchronization state. This test function requires an always activated receive strobe RSTRB_N



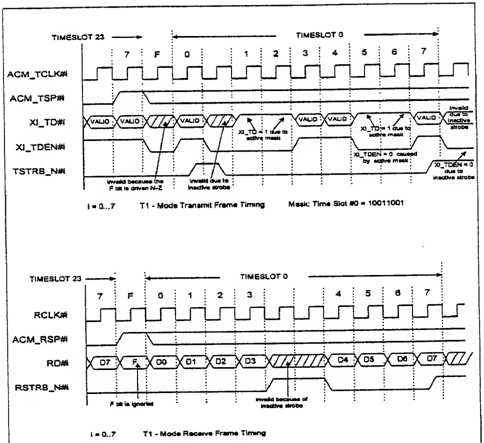
i = 0..27 E1 - Mode Transmit Frame Timing

i = 0..27 E1 - Mode Receive Frame Timing

PMR: PCM="1000" RXF='0' TXR='0'
CONF1: GPM = '0'

Figure 3.1.8.d

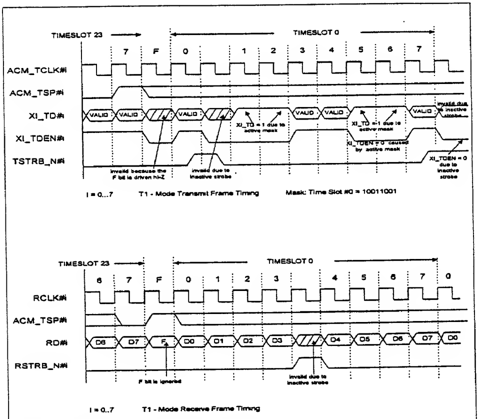
PCM interface timing for SCM: E1 with internal framing function (PMR.LT = 0, PMR.CTFSD = 0)


TIMESLOT 23
TIMESLOT 0
7
F
0
1
2
3
4
5
6
7
ACM_TCLK#i
ACM_TSP#i
XI_TD#i
XI_TDEN#i
TSTRB_N#i
VALID
XI_TD = 1 due to active mask
invalid due to inactive strobe
XI_TDEN = 0 caused by active mask
XI_TDEN = 0 due to inactive strobe
invalid because the F bit is driven hi-Z
invalid due to inactive strobe
i = 0...7
T1 - Mode Transmit Frame Timing
Mask: Time Slot #0 = 10011001
RCLK#i
ACM_RSP#i
RD#i
RSTRB_N#i
D7
F
D0
D1
D2
D3
D4
D5
D6
F bit is ignored
invalid because of inactive strobe
i = 0..7
T1 - Mode Receive Frame Timing

PMR: PCM="0000", TXR='0', RXF='0' , TBS = "100", RBS = "100", TSF = '0', RSF = '0'
CONF1.GPM = '1'

Figure 3.1.8e:

PCM interface timing for ACM: T1 without internal framing function
TBS = RBS = "100" corresponds to bit shift = 0 (s. PMR register)


TIMESLOT 23
TIMESLOT 0
7
F
0
1
2
3
4
5
6
7
ACM_TCLK#i
ACM_TSP#i
XI_TD#i
XI_TDEN#i
TSTRB_N#i
VALID
XI_TD = 1 due to active mask
XI_TD =1 due to active mask
invalid due to inactive strobe
XI_TDEN = 0 caused by active mask
XI_TDEN = 0 due to inactive strobe
invalid because the F bit is driven hi-Z
invalid due to inactive strobe
I = 0...7
T1 - Mode Transmit Frame Timing
Mask: Time Slot #0 = 10011001
TIMESLOT 23
TIMESLOT 0
6
7
F
0
1
2
3
4
5
6
7
0
RCLK#i
ACM_TSP#i
RD#i
RSTRB_N#i
D6
D7
F
D0
D1
D2
D3
D4
D5
D6
D7
D0
F bit is ignored
invalid due to inactive strobe
I = 0..7
T1 - Mode Receive Frame Timing

PMR: PCM="0000", TXR='1', RXF='1' , TBS = "100", RBS = "100", TSF = '1', RSF = '1'
CONF1.GPM = '1'

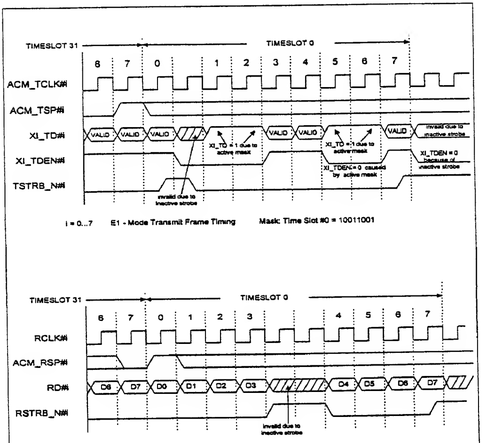
Figure 3.1.8f

PCM interface timing for ACM: T1 without internal framing function

Port mode register (PMR): RXF = 1, TXR = 1

Transmit: generation of transmit data at rising ACM_TCLK edge, sampling of the transmit sync pulse ACM_TSP at falling edge, sampling of the transmit strobe TSTRB_N at rising edge (TSTRB_N is expected to be provided half a clock cycle before the corresponding data are generated).

Receive: sampling of the receive data, receive sync pulse and receive strobe at falling RCLK edge.

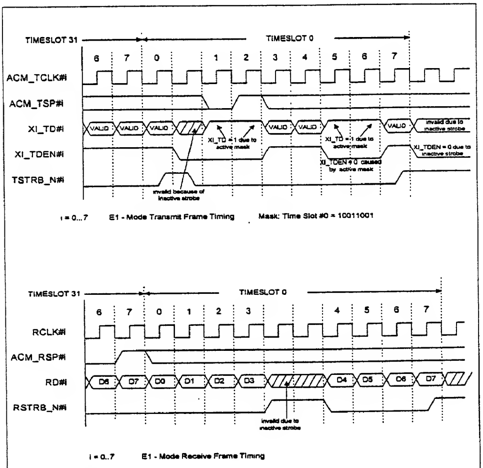

TIMESLOT 31
TIMESLOT 0
6
7
0
1
2
3
4
5
6
7
ACM_TCLK#i
ACM_TSP#i
XI_TD#i
XI_TDEN#i
TSTRB_N#i
VALID
VALID
VALID
XI_TD = 1 due to active mask
VALID
VALID
XI_TD = 1 due to active mask
VALID
invalid due to inactive strobe
XI_TDEN = 0 because of inactive strobe
XI_TDEN = 0 caused by active mask
invalid due to inactive strobe
TIMESLOT 31
TIMESLOT 0
6
7
0
1
2
3
4
5
6
7
RCLK#i
ACM_RSP#i
RD#i
RSTRB_N#i
D6
D7
D0
D1
D2
D3
D4
D5
D6
D7
invalid due to inactive strobe

i = 0...7 E1 - Mode Transmit Frame Timing Mask: Time Slot #0 = 10011001

i = 0..7 E1 - Mode Receive Frame Timing

PMR: PCM="1000", TXR='0', RXF='0' , TBS = "100", RBS = "100", TSF = '0', RSF = '0'
CONF1.GPM = '1'

Figure 3.1.8.g
PCM interface timing for ACM: E1 without internal framing function



TIMESLOT 31
TIMESLOT 0
6
7
0
1
2
3
4
5
6
7
ACM_TCLK#i
ACM_TSP#i
XI_TD#i
XI_TDEN#i
TSTRB_N#i
VALID
VALID
VALID
VALID
VALID
VALID
XI_TD = 1 due to active mask
XI_TD = 1 due to active mask
invalid due to inactive strobe
XI_TDEN = 0 due to inactive strobe
XI_TDEN ≠ 0 caused by active mask
invalid because of inactive strobe
TIMESLOT 31
TIMESLOT 0
6
7
0
1
2
3
4
5
6
7
RCLK#i
ACM_RSP#i
RD#i
RSTRB_N#i
D6
D7
D0
D1
D2
D3
D4
D5
D6
D7
invalid due to inactive strobe

i = 0...7 E1 - Mode Transmit Frame Timing Mask: Time Slot #0 = 10011001

i = 0..7 E1 - Mode Receive Frame Timing

PMR: PCM="1000", TXR='0', RXF='0' , TBS = "001", RBS = "101", TSF = '0', RSF = '0'
CONF1.GPM = '1'

Figure 3.1.8.h

PCM interface timing for ACM: E1 without internal framing function

TBS = "001" corresponds to bit shift = -3 of the transmit data relative to the transmit sync pulse ACM_TSP

RBS = "101" corresponds to bit shift = +1 of the receive data relative to the receive sync pulse ACM_RSP

4 Register Description

4.1 Register Overview

Register Overview Table : XPI V2.1

Register ID	Access	Absolute Address cs_n & a(7:2)	Reset Value	Comment
CONF1	R/W	40_H	00000000_H	(virtual global) configuration register 1
CONF2	R/W	44_H	00000000_H	(virtual global) configuration register 2
PMIAR	W	60_H	00000000_H	(macro specific) port mode indirect access register
PMR	R/W	64_H	$0104C000_H$	(macro specific) port mode register
REN	R/W	68_H	00000000_H	(macro specific) receive enable register
TEN	R/W	$6C_H$	00000000_H	(macro specific) transmit enable register

4.2 Detailed Register Description

4.2.1 (Virtual Global) Configuration Register 1(CONF1)

Access	: read/write
Address	: 00000040_H
Reset Value	: 00000000_H

31															16

15															0
															GPM

Note: *This register is common for XPI and other macros. The macro XI just collect those data bits from the written dword (see above), that are relevant for macro XI. With a read access to this register, the macro XI drive only the value for the relevant bits. All other bits are set to '0'.*

GPM: general PCM port mode:
'0' : standard channelized mode (SCM): 28 * T1 / E1 lines with chip internal framing function
'1': alternate channelized mode (ACM): 16* T1 / E1 lines lines (port 0...15) without chip internal framing function

Please note: M256_MODE = "00" forces the M256F in ACM mode

4.2.2 (Virtual Global) Configuration Register 2 (CONF2)

Access : read/write
Address : 00000044_H
Reset Value : 00000000_H

					26 24			21	16
					PORTQ[2:0]			RSP EN	SPAD[4:0]

		13	12 8								
		ILP	LPPID[4:0]								

Note: *This register is common for XPI and other macros. The macro XI just collect those data bits from the written dword (see above), that are relevant for macro XI. With a read access to this register, the macro XI drive only the value for the relevant bits. All other bits are set to '0'.*

LPPID[4:0]: port selection for internal portwise loop

ILP: internal portwise loop on port selected by LPPID[4:0]

SPAD[4:0] port selection for signaling the receive sync information provided by the on chip framer on pin RSPO_TCLKO (relevant in SCM mode only)

RSPEN: if set to '1', the regenerated framer sync pulse is visible on RSPO_TCLKO
if reset to '0', the effective transmit clock of port#0 is visible on RSPO_TCLKO

PORTQ[2:0]: port interrupts will be written into this queue

4.2.3 Port Mode Indirect Access Register (PMIAR)

Access	: write
Address	: 00000060$_H$
Reset Value	: 00000000$_H$

31								23							16
								AI							

15											4				0
											PORT(4:0)				

Note: This register is implemented once for all ports in the XPI

PORT[4:0]: Port number, to/from which the data in register PMR is written/read

AI: autoincrement mode, if set to '1' when writing the PMIAR, the port number bit field is automatically incremented after each write or read access to the data register PMR. The port number will wrap automatically at the maximum port number (M256F: port# 27)

4.2.4 Port Mode Register (PMR)

Access	: read/write
Address	: 00000064_H
Reset Value	: $0104C000_H$

31 28				ACM				ACM
PCM[3:0]	0	0	0	TBS[2:0]	0	0	0	RBS[2:0]

15				ACM	ACM			7							0
REIM	TEIM	RXF	TXR	RSF	TSF	CTFSD	LT	0	0	0	0	0	0	0	0

Note: This register is available for each port (register bits labeled by "ACM" exist physically only for ports0...15, in contrast to the remaining bits that exist for each of the 28 ports)

PCM(3:0):

0000	T1 / 1.544MHz
1000	E1 / 2.048MHz
1111	Unchannelized

TSF, RSF
'0': sample TSP/RSP at rising edge
'1': sample TSP/RSP at falling edge
(don't care for SCM)

TXR
'0': transmitting TX data at falling edge, sampling TSTRB_N at falling edge, sampling CTFS in SCM mode at rising edge
'1': transmitting TX data at rising edge, sampling TSTRB_N at rising

edge, sampling CTFS in SCM mode at falling edge
(TXR must be programmable also for SCM !)

RXF ´0´: sampling RX data and TSTBR_N at rising edge
´1´: sampling RX data and RSTRB_N at falling edge
(RXF must be programmable also for SCM !)

TEIM: transmit sync Error Interrupt Mask (reset value = '1')

REIM: receive sync Error Interrupt Mask (reset value = '1')

TBS, RBS : bit shift offset of +3bits/- 4bits **(don't care for SCM)**
Note: If an internal loop is active, RBS must be identical to TBS!!

Bit24	Bit23	Bit22	shift of TD bit relative to TSP	Bit18	Bit17	Bit16	shift of RD bit relative to RSP
0	0	0	-4	0	0	0	-4
0	0	1	-3	0	0	1	-3
0	1	0	-2	0	1	0	-2
0	1	1	-1	0	1	1	-1
1	0	0	0	1	0	0	0
1	0	1	1	1	0	1	1
1	1	0	2	1	1	0	2
1	1	1	3	1	1	1	3

LT: if set to '1', the clock source of the serial transmit part is switched to RCLK

CTFSD: if set to '1', CTFS is ignored as external synchronization signal for transmit frames in SCM mode

4.2.5 Receive Enable Register (REN)

Access	: read/write
Address	: 00000068_H
Reset Value	: 00000000_H

31				27											16
				REN 27											

15															0
															REN0

REN[27:0]: if set to '1', the corresponding receive port is enabled, i.e. incoming data is processed. After synchronization is achieved, the XPI requests service from the TSAR for further data processing.
If set to '0', no requests are forwarded to FRAMR and TSAR.

4.2.6 Transmit Enable Register (TEN)

Access	: read/write
Address	: $0000006C_H$
Reset Value	: 00000000_H

31				27											16
				TEN 27											

15															0
															TEN0

TEN[27:0]: if set to '1', the corresponding transmit port is enabled, i. e. sends data originating from PMT if the corresponding timeslot is also enabled in TSAT.

If set to '0', the corresponding output port is Hi-Z generally and no requests are forwarded to FRAMT and TSAT.

Appendix O

Title:
Platform Concept:
SMIF Specification
Version 1.0

0. Abbreviations

PP	Platform peripheral
ADCB	Address/data/control bus
Word	32 bits
Halfword	16 bits
Byte	8 bits
STA	Static timing analysis
EBI	External bus interface
BPI	Platform bus/peripheral interface

1. Introduction

Please note that all following information applies to fully digital, synthesizable Platform peripherals. Even in analog modules there is at least one digital block containing the SFR file, bus interface and control logic. The information given here is also valid for these digital parts.
Fig. 1.1 shows the PP development flow.
All PPs are first described by a C-model coming from Product Definition, and a configurable version of a synthesizable VHDL model. We call this configurable version "Softmacro", because it offers the possibility to select among several bus models where this peripheral shall be connected to, and configure the SFR address space.

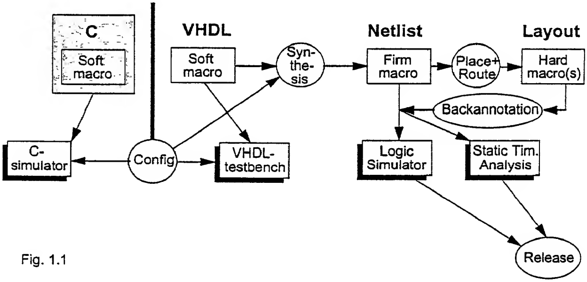
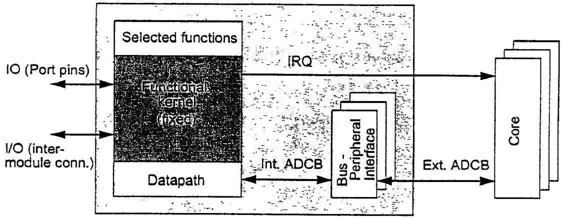


Fig. 1.1

To simulate the peripheral's behavior, the softmacro has to be configured:
one bus model has to be selected,
the SFR configuration has to be defined.
As soon as the configuration parameters have been filled out, the C/VHDL description is updated by a script. The updated C/VHDL file can be used for running functional simulations controlled by a C/VHDL testbench. The testbench is able to simulate the module's functionality by means of connecting the selected bus model.
Additionally, it can be synthesized into a firmmacro. The firmmacro is a netlist view, containing the gate-level representation of the PP. The firmmacro is simulated using a netlist-based sim-

ulator. Whereas the functional characterization took place on the softmacro C/VHDL base, the timing verification is performed on the firmmacro by using STA and backannotation (pre- and post-layout). When the timing characterization results meet the specification, the PP is released into the Platform library.



□ Complete softmacro
▨ Fixed functional kernel
□ Configurable parts

Fig. 1.2

2. Softmacro parts

Fig. 1.2 shows the view of a Platform peripheral.
Dark shaded is the fixed functional kernel which is not intended to be changed during the peripheral's lifetime. White fields indicate configurable parts of the peripheral. These are:

additional/selected function blocks,
SFR datapath,
bus bridge to core.

The *light shaded* area indicates the complete softmacro. The C/VHDL files are configured by generics and constants set in packages.
The VHDL files can be synthesized into a firmmacro. The firmmacro is a netlist view, containing the gate-level representation of the PP. The firmmacro is simulated using a netlist-based simulator. Whereas the functional characterization took place on the softmacro C/VHDL base, the timing verification is performed on the firmmacro by using STA and backannotation (pre- and post-layout).

2.1. Additional/selected function blocks

They may or may not be needed in the peripheral.
Example of an additional function is a second serial communications channel. In this case, the channel's function is only described once, and if desired in the PP, it is doubled by a C/VHDL

configuration parameter.
Example of a selected function is having a timer overflow signal either high-active or low-active. In this case both versions are part of the fixed functional kernel, and the selection is made by connecting a control input signal to either VDD or VSS („Hardware switch").
Both the additional and the selected functions are configured by one or more configuration parameters which are part of the C/VHDL softmacro.

2.2. The complete datapath containing all SFR registers

The datapath width is selected automatically from the desired databus. If FPI32 is selected, all SFRs have 32 bits. For lower-performance peripherals a 16bit-wide FPI16 is defined. It is actually a part of FPI32 which is seen by a peripheral with halfword SFRs. With the Platform definition of supported busses (see section 4.2.), it is possible to:

use byte-oriented cores with byte-oriented peripherals only,
use halfword-oriented cores with halfword-oriented peripherals only,
use word-oriented cores with halfword-oriented peripherals,
use word-oriented cores with word-oriented peripherals.

2.3. Bus - Peripheral Interface (BPI)

This is the interface between the external address/data/control bus (Ext. ADCB) and the internal ADCB. Although the external data and address bus may be tristate, the internal bus system must not be tristate. Therefore separate input and output data busses are generated inside the peripherals. This is a prerequisite for performing true STA.

3. Module interface

As can be seen in Fig. 1.2, we have several groups of I/O pins which connect the peripheral to its environment:
a) I/O from/to port pins (special product-related I/O functions),
b) I/O from/to other on-chip peripherals or processor core,
c) I/O from/to on-chip ADCB,
d) interrupt request outputs.

3.1. Structural test coverage

With respect to the Platform's "Design for Test (DFT)" concept, all digital Platform peripherals will contain one or more *full scan chains* to achieve a close-to 100% structural test coverage on silicon. The required test patterns are generated on chip level via ATPG. There will be as many parallel short scan chains as can be accepted due to the number of product port pins. Every scan chain occupies two port pins (scan_in, scan_out). Scan tests are controlled by a centralized Platform module containing a JTAG-like TAP-controller. It is unique for all Platform products. For more details on the TAP controller refer to the document „dft.fm5".

3.2. Functional test coverage

During peripheral development and for the silicon's AC verification, *functional patterns* have to be provided to check the peripheral's behavior. To be independent of the product's set of peripherals, all functional tests are strictly peripheral-oriented and stimulate directly the peripheral's interface input signals. This strategy is known as *module isolation*. Only one PP can be in „module isolation" testmode at one time. If one PP is in module isolation testmode, all other PPs and the core must stay in „quiet mode", i.e. must release the ADCB and must not draw any static current. PP-related IDDQ tests are also performed using module isolation. It is therefore assumed that on product level, all the peripheral's I/O pins can be accessed directly from external with the exception that a pad input/output driver is interconnected.
This assumption is of course valid for all "I/Os from/to port pins".
All inputs coming from other on-chip modules have to be multiplexed on additional port pins.
The ADCB is expected to be accessible via port pins, too (EBI assumed). The existence of an EBI is true for most of the existing (non-Platform) products. In the age of bigger on-chip memories, more and more future products will lack an EBI.
An additional exception arises with DPI since the external DPI (visible at the product pins) runs with reduced speed compared to the on-chip (internal) DPI. The peripheral itself will be connected to the internal DPI. Synchronization is maintained by the external bus controller (EBC). Additionally the number of externally visible address lines and control signals is reduced. An EBC model is required to be part of the C/VHDL testbench in order to translate the PDL patterns into external DPI bus signals.
All outputs which are not directly accessible, are connected to a special test circuit called MISR (multiple input shift register). This MISR generates a unique signature value during the functional tests. By reading the signature after test completion, one can tell if the tests passed or failed. It is even possible to output the MISR output signal during the test. This allows error detection earlier and simplifies error tracking.
However, these DFT-related guidelines can be found in the "Design for Test (DFT)" chapter.
Platform restricts interrupt behavior to a common strategy. All Platform peripherals output their interrupt request signals. An IRQ is indicated by a HIGH pulse which lasts at least one clock cycle. The IRQ flipflops which are set by those pulses are located in an interrupt controller or (like C166 and Dolphin) in several service request nodes (SRN).

4. Configuration parameters

All configuration parameters have to be filled out by the peripheral designer. Complete set of parameters allows synthesis into a firm macro.

4.1. SFR configuration

Configuration of the special function register file is controlled by a list of SFRs, containing SFR name, address, physically available bits (e.g. [31:0], [31:16,7:0]).
Bus and datapath width:
SFRs have by default the same data width as the bus bridge width. The bus bridge interfaces the PP's SFRs to the inter-module ADCB.
Not all of the bits need to be physically available, i.e. readable or writable.
The SFR address given in the configuration list is always a byte address.
In a 16bit-peripheral SFR addresses have the least significant bit=0. Two valid-byte signals allow lower and upper byte access on a halfword-wide SFR. Halfword access is restricted to addresses with least significant bit=0.
In a 32bit-peripheral SFR addresses have the least two significant bits=00. Four valid-byte signals allow byte access and halfword access on a word-wide SFR. Byte access can take place for any of the four bytes, halfword access can take place for lower and upper halfword of the

word. Word access is restricted to addresses with least significant two bits=00; halfword access is restricted to addresses with least significant bit=0.

4.2. Bus configuration

The Bus - Peripheral Interface (BPI) is specific for the target bus the PP is intended to be connected to. The softmacro does not imply a specific BPI. For simulation with the C/VHDL testbench, the target bus has to be selected. BPI serves as an interface between the PP's SFRs and other on-chip modules which are connected to the same ADCB.

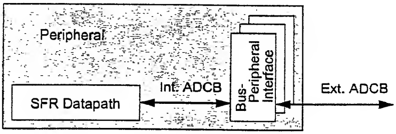


Fig. 4.2

4.2.1. External ADCB signals

Basically every ADCB supported by Platform contains the following signals. „ADCB“ is replaced by the bus identifier. For final names refer to chapter 5 of this document.

	ADCB_D[n-1:0]	Data bus (I/O), width n;
	ADCB_A[m-1:0]	Address bus (I), width m;
	ADCB_WR	Write SFR (I), active high;
or	ADCB_WR_N	Write SFR (I), active low;
	ADCB_RD	Read SFR (I), active high;
or	ADCB_RD_N	Read SFR (I), active low;

Address/data bus may be multiplexed. In this case an address latch enable (ALE) signal is required.

Optional additional control signals with not predefined names serve as:

Peripheral select (I);
Ready (I/O), wait-state support;
Alive (I/O), indicates that PP address is met;
Opcode (I), select data transfer width and protocol.

The set of signals for every Platform-supported bus is listed in chapter 5 of this document.

4.2.2. Internal ADCB signals

Whereas the set of external control signals differs among the ADCBs, the internal SFR control signals are common independent of the target ADCB. The internal control signals are listed below. Note that the direction is given from view of the bus interface block.

Mandatory signals:

BPI_DATA_I[n-1:0]	Input data bus (I), width n;
BPI_DATA_O[n-1:0]	Output data bus (O), width n;
BPI_WR_SFRx_N	Write data to SFR number x (O), active low;
BPI_RD_SFRx_N	Read data from SFR number x (O), active low;

Optional signals:

BPI_WR_BY_N[k:0]	Select SFR bytes for write operation (O), active low, k=1 for 16bit PP, k=3 for 32bit PP;
BPI_RDWR_N	Read-modify-write indication (O), active low.
PER_RDY	Ready signal of peripheral kernel

There are two reasons for separating the data input and output busses:
1) a tristate bus complicates static timing analysis, ATPG, synthesis, etc.
2) since a read operation takes place in cycle #n, and a write operation takes place in cycle #n+1, bus contention would appear if read and write data shared the same bus; see fig. 4.2.2.

Fig. 4.2.2.1 shows a sequence of „read SFR data #1 / write SFR data #2 / read SFR data #3". As an example the signal timing is taken from PPI bus:
Cycle #1 rising CLK drives address for read data #1 on external bus ADCB_A and activates internal read signal BPI_RD_SFRx_N; the read SFR data #1 is expected to appear on internal data bus BPI_DATA_I before cycle #2 rising CLK.
Cycle #2 rising CLK drives address for write data #2 on external bus ADCB_A, and drives read data #1 on external bus ADCB_D. Simultaneously, the write data #2 is announced to the bus interface, but is actually driven one cycle later (DPI bus pipelining).
Cycle #3 rising CLK address for read data #3 on external bus ADCB_A, and drives write data #2 on external bus ADCB_D. This data is immediately propagated onto the internal data bus BPI_DATA_O, and write signal BPI_WR_SFRx_N is active. Simultaneously, internal read signal BPI_RD_SFRx_N is activated; the read SFR data #3 is expected to appear on internal data bus BPI_DATA_I before cycle #4 rising CLK.
In cycle #3 internal read and write signals are simultaneously active, and read and write data are driven simultaneously on the internal I- and O-busses, respectively.

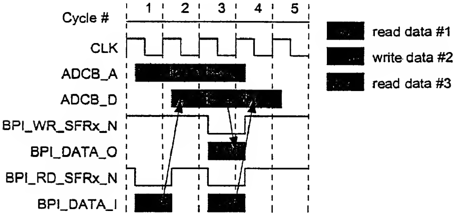


Fig. 4.2.2.1

Fig. 4.2.2.2 shows a sample BPI for an external Platform bus interfacing to a passive slave PP. Note that two additional signals connect BPI and peripheral kernel. Signal BPI_RMODW_N indicates to the peripheral kernel that a read-modify-write access is in progress. Signal

PER_RDY is needed if a peripheral is slower than the bus. This signal indicates when the peripheral has finished with the access.

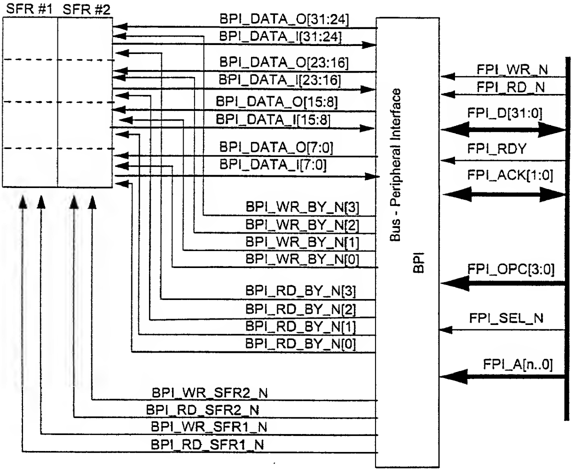


Fig. 4.2.2.2

4.2.3. SFR data widths

Not necessarily, the ABCD bus width (i.e., the width of the master) is the same as the width of a passive slave PP. This gives us the following scenarios:

Tabelle 1:

ADCB data width	SFR register width	SFR access width	Comment
32	32	32	word access
32	32	16	half word access
32	32	8	byte access
32	16	16	word or half word access
32	16	8	byte access
32	8	8	any width for access
16	32	16	sel.halfword by addr.
16	16	16	standard
8	32	8	sel. byte by addr.
8	16	8	sel. byte by addr.
8	8	8	standard

For a 32 bit bus, each SFR address must be mapped to a 32 bit address regardless whether the SFR width is 8, 16, or 32 bits. (This means for 32FPI, that each SFR address ends on 00 even if the SFR is only 8 bit wide.) Similarly for a 16 bit bus, each SFR address of an SFR with width 8 or 16 bit must be mapped to a 16 bit address.
Data is shown on the bus at the location where it resides in the SFR. If for example the second byte of a 32 bit register is read, this byte is visible on the second byte of the bus as well.
To make the scenarios clearer, let us pick examples for the FPI bus:

If the 32bit SFR shall be written to an 8bit ADCB, the data byte has to be directed to its proper target location. This is done by the internal control lines BPI_WR_BY_N[3:0]. They have the same timing as the write signals BPI_WR_SFRx_N. Note that a problem will arise here if read a continuously changing 32 bit SFR (e.g., a 32 bit counter value). Since such a value will have changed until the next byte is read, we have to think of implementing a 32 bit buffer register that stores the value when the first byte is read. And when the next byte is read in a later cycle, it is read from this buffer register.
Similar strategies apply to a 16bit PP connected to an 8bit ADCB and to a 32bit PP connected to a 16bit ADCB. The byte and halfword data are expected to appear on their corresponding positions on the external ADCB. Fig. 4.2.3 shows the mapping of SFR word/halfword/byte addresses to the internal BPI_WR_BY_N[3:0] lines.

For read access, usually the complete 32 bits are read even if only a byte access occurred. This is the easiest way but may cause trouble for destructive read (an SFR changes its value as soon as it is read). For this case, also signals BPI_RD_BY_N[3:0] are provided by BPI and can be used by the peripheral kernel optionally.

PPs with more SFR bits can be connected to ADCBs with less data bits; e.g. a 32bit timer can be connected to PB8. In this case, since only one byte can be accessed at one time, the Platform standard says:
A 32bit SFR is built from 4 consecutive bytes. The basic SFR address is ...xx00. This is also

the address of its least significant byte. The SFR contains also bytes ...xx01, ...xx10 and ...xx11. Byte ...xx11 is the most significant byte.
A read access on SFR byte ...xx00 loads the complete 32bit value of the SFR into a buffer. The other 3 bytes have to be read from this buffer consecutively by applying addresses ...xx01, ...xx10 and ...xx11 in any order. *Important is that any access to byte ...yy00 of another SFR will immediately overwrite the current buffer contents.*

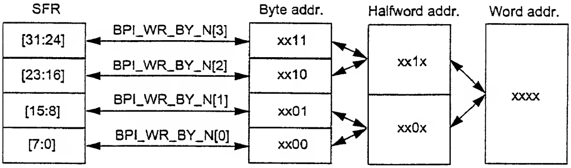


Fig. 4.2.3

Please note again that an SFR read access does always read the complete SFR, even if FPI_OPC[3:0] denotes a halfword or byte access. Halfword and byte access is only evaluated for write accesses.

4.3. Functional configuration

This feature to add or select functionality is optional. If some functional subblocks shall be omitted or exchanged, VHDL models of each subblock must be provided. According to the value of the functional parameters, the corresponding subblocks' VHDL models are merged into the softmacro description.
However, the functional kernel of a Platform peripheral will never be changed. Functional configuration parameters are needed only to add functionality to the kernel or enable/disable certain functions within the kernel.

4.4. BPI architecture

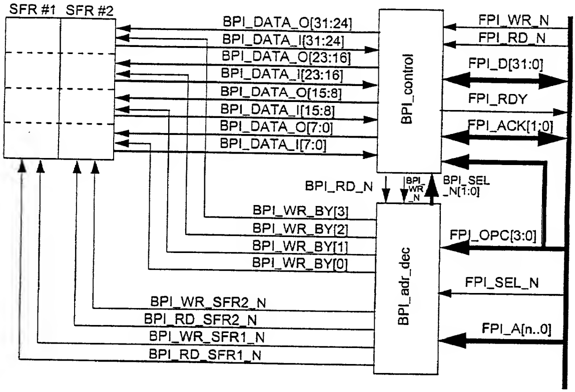


Fig. 4.4.

Fig. 4.4. illustrates the BPI architecture for the example of an external Platform bus interfacing to a passive slave PP. Signals BPI_RD_BY[3:0] are omitted in Fig. 4.4 for sake of simplicity. Basically, BPI consists of two parts, the BPI control unit (BPI_control) and the BPI address decode unit (BPI_adr_dec).
BPI_control connects external and internal data busses, passes read/write information to BPI_adr_dec, and drives other control information such as RDY (ready) and ACK (acknowlegde).
BPI_adr_dec decodes the external address bus and evaluates which SFR is accessed. It ands this information with the read/write information from BPI_control and drives the SFR-specific read/write lines (BPI_RD_SFRx_N, BPI_WR_SFRx_N). Simultaneously, BPI_adr_dec generates the byte-select signals for writing byte- or halfword-parts of a SFR from either FPI_OPC[3:0] or it may be hard coded that, e.g., only 16 bits accesses are possible if the external bus has 16 bit. In case of FPI, a pre-decoded select signal called FPI_SEL_N is provided by the central bus controller.
In case of a byte (or 16 bit) access to a wider SFR (32 bit SFR), there are two scenarios. Firstly, if a 32 bit bus is connected to BPI, the whole data bus is forwarded by the BPI_control. In case of a read access, the complete SFR is read. In case of a write access, the complete bus is forwarded to the internal bus and the peripheral kernel has to activate only the write lines of the selected bits. Secondly, if a bus smaller than 32 bits is connected. It must be provided that the selected byte of the SFR (e.g., the highest byte of the SFR) is shifted to a lower location so that it can be read by the external bus. The opposite shift operation is needed for write access.

5. Platform-supported ADCBs

This chapter describes all ADCBs which are supported by Platform. A complete list of signals is given, plus a connection block diagram and a timing diagram.

5.1. 8-bit peripheral busses

5.1.1. PB

PB_AD[7:0] Address/data multiplexed, address range 00h..FFh
(Note: 8051 architecture reserves addresses 00h..7Fh for on-chip RAM).
PB_ALE Address latch enable, active high (was originally LDIRA_B),
PB_RD_N Read SFR data, active low (was originally RDIRA_B),
PB_WR_N Write SFR data, active low (was originally WDIRA_B).

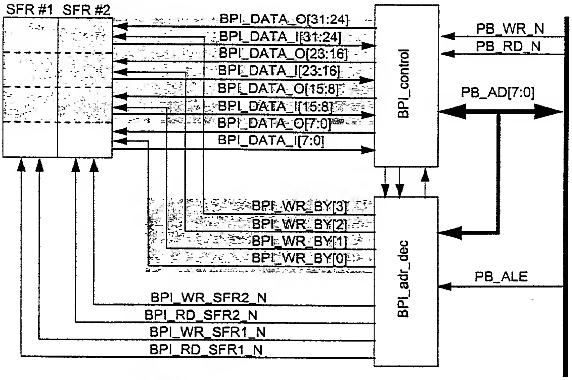


Fig. 5.1.1.1. ▨ mean optional architectural parts

Figure 5.1.1.2 illustrates read and write access. Let SRF1 have address adr1 and SFR2 adr2.

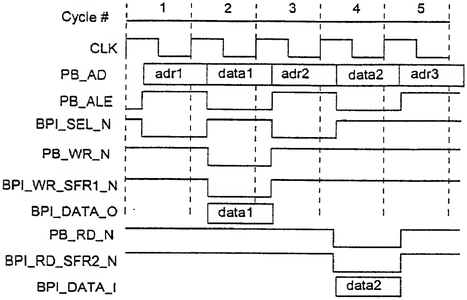


Fig. 5.1.1.2

BPI_adr_dec decodes PB_AD during PB_ALE. With rising clock edge in the beginning of cycle 2, the coming access to SFR1 is detected. With PB_WR_N going low, BPI_WR_SFR1_N is driven low so that SFR1 can be set to the value on BPI_DATA_O with the next rising clock edge. Also, BPI_adr_dec drives BPI_SEL_N active so that BPI_control drives data1 onto BPI_DATA_O. A similar scenario applies to the read access in the next two cycles. The address adr3 is not an address of any SFR in the considered peripheral so that BPI_SEL_N stays inactive.

Contrary to Figure 5.1.2, the current implementation of PB always precharges PB during CLK='1'. Thus, all reads and writes can only be active during CLK='0'. However, this is subject to change for an 8 bit platform core and thus not considered here.

5.1.2. MB

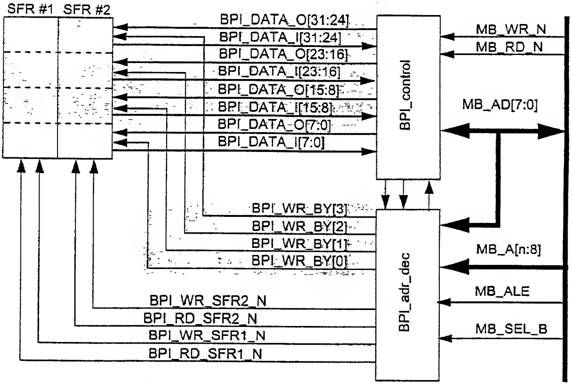


Fig. 5.1.2.1. ▨ mean optional architectural parts

MB replaces the original AMO-XDB. To avoid naming conflicts with the 16bit XBUS (=XB in Platform), XDB has been renamed to MB (Memory bus).

MB_AD[7:0] Low address/data multiplexed,
MB_A[15:8] High address byte, address range 0000h..FFFFh
Note: not all most significant address lines need to be connected.
MB_SEL_B PP select, active low (was originally CSCODE_B or CS_XRAM_B),
MB_ALE Address latch enable, active high (was originally XDBALE),
MB_RD_N Read SFR data, active low (was originally RD_B),
MB_WR_N Write SFR data, active low (was originally WR_B).

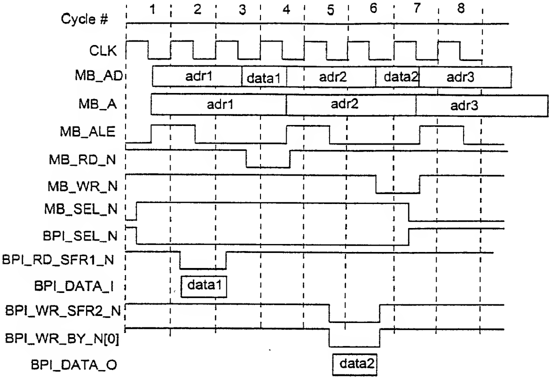


Fig. 5.1.2.2.

Figure 5.1.2.2. shows a read and a write access with MB. The exact timing is not yet specified for a new-to-develop 8 bit platform core.

5.2. 16-bit peripheral busses

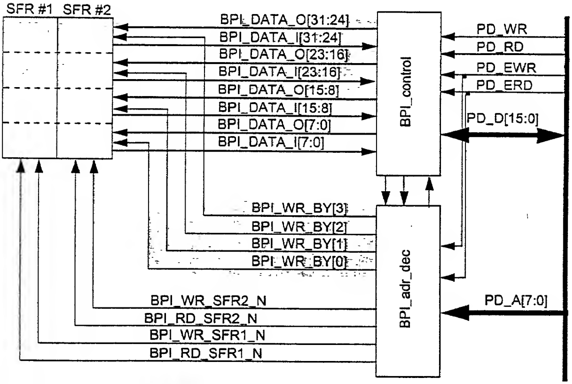


Fig. 5.2.1.1. ⋯ mean optional architectural parts

5.2.1. PD

PD_D[15:0] Data (was originally PD[15:0]),
PD_A[7:0] Address, range 00h..FFh (was originally PA[7:0]),
PD_RD Read SFR data, active high (was originally DW_PRD),
PD_WR Write SFR data, active high (was originally DW_PWR),
PD_ERD Read extended SFR data, active high (was originally DW_EPRD),
PD_EWR Write extended SFR data, active high (was originally DW_EPWR).
Note: PD_RD/PD/WR and PD_ERD/PD_EWR select different 256 SFR blocks, thus a total of 512 SFRs are available.

Figure 5.2.1.1. shows PD_EWR and PD_ERD be connected to BPI_adr_dec. This is because extended read or write is actually an SFR extension, i.e., if PD_EWR is used instead of PD_WR, a different SFR is accessed with the same address.

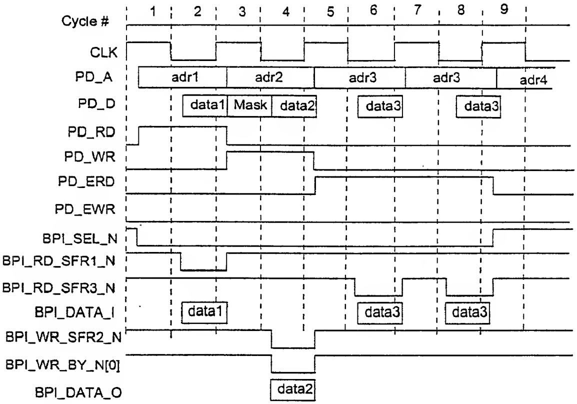


Fig. 5.2.1.2.

Figure 5.2.1.2. illustrates a normal read, a normal write access and an extended read access. Note that setup and hold times of the PD bus are defined in the PD bus specification (PD_Bus Specification V0.2, Axel Freiwald). These times have to be met in order to allow connecting BPI to the PD in an old device. Thus, VHDL description and synthesis has to meet be constrained carefully and also simulation after synthesis (and layout) has to be carried out thoroughly.

5.2.2. XB

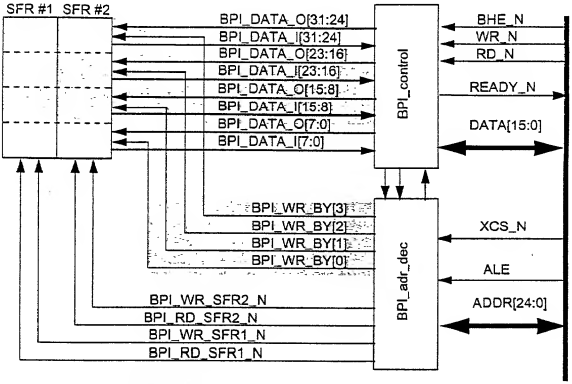


Fig. 5.2.2.1. ▨ mean optional architectural parts

XB replaces the original XBUS.

The XB is not very nice to implement in a synchronous environment due to its timing specification. This holds for the synchronous mode and is even worse for the asynchronous mode. Therefore, a BPI for XB will hardly be implemented in a strict synchronous fashion. This induces that significant problems will arise if static timing analyses, ATPG, etc. shall be applied. Nevertheless, it is feasible to develop a BPI even for the XB asynchronous mode. Thus, here are some basic ideas.

XBUS_data[15:0]	data,
XBUS_addr[23:0]	address, address range 000000h..FFFFFFh (16 Mbytes)

Control lines (CS_B, RD_B, WR_B, READY, RST), support of several bus protocols incl. wait states.

Synchronous mode:

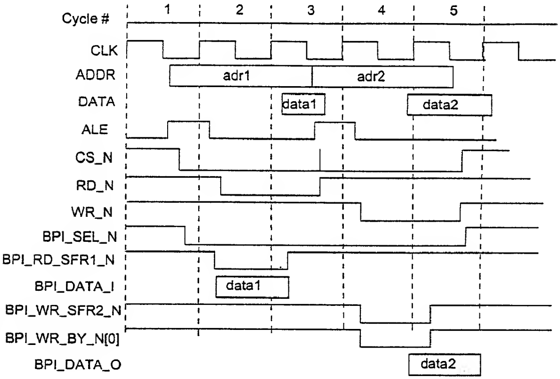


Fig. 5.2.2.2.

The general scheme for normal read and write access can be seen in Figure 5.2.2.2. Note that the timing specification of the XBUS has to be met.

Let us hint on two unpleasant features. For read access, data is never driven during a rising clock edge but around a falling edge (see data1 on signals DATA). Thus, driving data onto the DATA bus must be carried out by some not strictly synchronous logic. For write access, the data (data2 on DATA) are valid only very shortly before the rising clock edge. So, there might not be enough time to write this data to a SFR with the rising clock edge. If this is the case, data should be written with the next rising clock edge (beginning of cycle 6 in Figure 5.2.2.2.). If this is the case, some changes are necessary to Figure 5.2.2.2: Signal BPI_SEL_N must be prolonged by one clock cycle, BPI_WR_SFR2_n and BPI_WR_BY_N[3:0] are valid one clock cycle later, and data2 will be seen on DPI_DATA_I one clock cycle later.

BPI_adr_dec should store the address on ADDR with the rising clock edge if ALE is high and CS_N is low. Then the stored address has to be decoded.

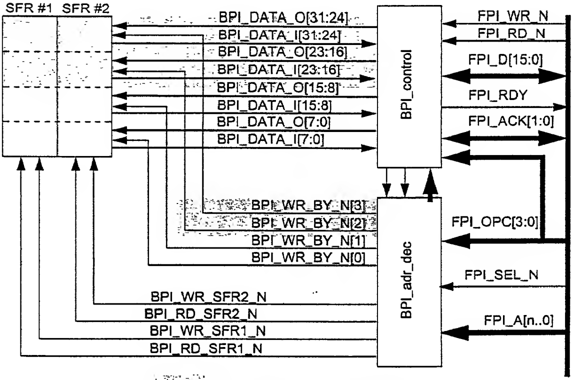
Asynchronous mode:

XB can also be used in an asynchronous mode. Then, all timings are related to the falling edge of ALE. No relationship to a clock is mandatory. We may want to proceed as follows. For read access, BPI_RD_SFRx_N can be activated as soon as the address is recognized to be valid, which is at the falling edge of ALE. To drive data with the correct timing onto bus DATA, some asynchronous logic will be necessary. For write access, BPI_WR_SFRx_N and BPI_WR_BY_N should be activated as soon as the data is available. If now read or write access needs more time because for example the XB is clocked faster than the peripheral, wait statements can be inserted by a peripheral using READY_N.

Wait state insertion:

The duration between falling edge of ALE (i.e., when the address is valid) and the time when the data is or has to be valid can be extend by one or more clock cycle using the ready signal. Also for this feature, please refer to the XBUS specification.

5.2.3. FPI



▨ mean optional architectural parts

Fig. 5.2.3.1.

Mandatory signals:

FPI_D[15:0]	Data, only the lower 16 of the 32 FPI data lines are connected,
FPI_A[31:0]	Address, range 00000000h..FFFFFFFFh (4 Gbytes), not all address lines will be connected,
FPI_SEL_N	Chipselect, active low, might be common to several PPs,
FPI_RD_N	Read SFR data, active low,
FPI_WR_N	Write SFR data, active low,

FPI_RDY Ready signal, active high, indicates waitstates,
FPI_OPC[3:0] Opcode, indicates data width and data transfer protocol,
FPI_ACK[1:0] Slave response code.
FPI_RDY Ready signal from peripheral, active high, indicates waitstates if inactive

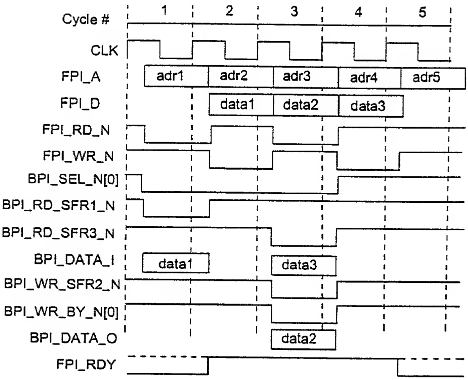


Fig. 5.2.3.2.

Figure 5.2.3.2 illustrates simple read and write accesses without wait states. FPI is pipelined so that the data that belong to an address appear one clock cycle later. For this reason, care has to be taken that read and write accesses and the data are forwarded at the right time. As indicated in Figure 5.2.3.2, write signals are delayed by one clock cycle while write data are forwarded immediately. For read access, we have a different story. The read signal is forwarded immediately so that also the read data are received by BPI immediately (immediately means in the same clock cycle). With the next rising clock edge, these data are driven onto FPI_D.

Note that an implementation problem might arise here. This is if decoding the address on FPI_A, switching BPI_RD_SFRx active, and returning the corresponding data to BPI takes more time than one clock cycle. If it turns out that this is not feasible in one clock cycle, then we must redefine BPI in a way that the peripheral kernel drives out data one clock cycle later and these data are then forwarded immediately to FPI_D by BPI.

Implementation hint: As FPI_SEL_N is decoded by a bus controller, this decoding will cause a delay on this signal. This means that the address on FPI_A is valid earlier than FPI_SEL_N. Thus, the address should be decoded first and determined valid or not with FPI_SEL_N, then.

Let us now discuss the purpose of signals BPI_SEL_N[1:0] (see also Figure 4.4, signals between BPI_control and BPI_adr_dec). Signal BPI_SEL_N[0] is the forward of signal

FPI_SEL_N. BPI_SEL_N[1] is activated if the address on FPI_A meets any SFR address of the peripheral kernel. These signals are necessary for BPI_control to be able to recognize whether a valid address is on FPI_A when FPI_SEL_N is activated. If this is the case, BPI_control must drive FPI_RDY active. Otherwise, BPI_control must drive an error message on FPI_ACK.

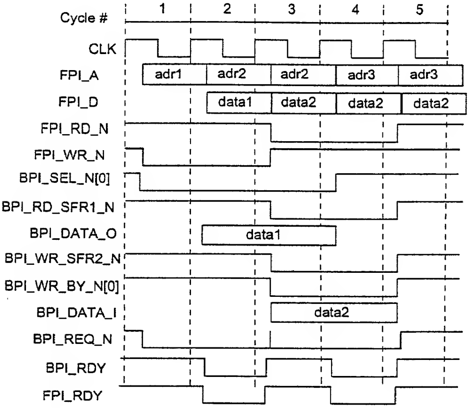


Fig. 5.2.3.3.

Figure 5.2.3.3. shows BPI connected to a peripheral that inserts one wait state for each access. This can be necessary for a slow RAM or a peripheral with a slow clock.
Similar as shown in Figure 5.2.3.2, the write signal is delayed by one clock signal. For read, the read data are delayed by one clock signal.

The number on inserted wait states is flexible. It depends on when the BPI_RDY signal from the peripheral kernel is active. This leaves the full flexibility of how many wait states must be inserted to the peripheral kernel. However, at least one wait state is required then. This restriction is necessary since we need one wait state to make RDY signal handshake.

If a write is followed by a read access, two different implementations of BPI can be selected that avoid that the old data is read. First alternative inserts an additional wait state. Second alternative provides multiplexers to forward data.

5.3. 32-bit peripheral busses

5.3.1 FPI

Signals:

FPI_D[31:0]	Data,
FPI_A[31:0]	Address, range 00000000h..FFFFFFFFh (4 Gbytes), not all address lines will be connected,
FPI_SEL_N	Chipselect, active low,
FPI_RD_N	Read SFR data, active low,
FPI_WR_N	Write SFR data, active low,
FPI_RDY	Ready signal, active high, indicates waitstates,
FPI_OPC[3:0]	Opcode, indicates data width and data transfer protocol,
FPI_ACK[1:0]	Slave response code.

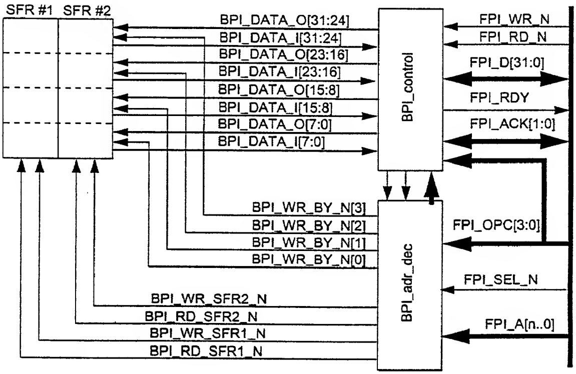


Fig. 5.3.1

Note: The number of control lines and thus the complexity of the bus interface grows with the required functionality: passive slave is easiest, intelligent master is most complex. BPI supports only passive slaves.

6. Bit Access

A bit write works as follows. The CPU performs consecutively a read and a write. For FPI bus, where a multi master concept applies, a read modify write access is performed which induces that the bus is locked. The whole read SFR is modified only in the bit that has to be changed. The whole value of the SFR is then written to the SFR.

Bit protection mode:
If a bit is bit protected, we must take special care. Let us assume a write bit access is performed by read modify write. In the affected SFR, also a bit protected bit exists. Now imagine that during the write access due to the read modify write, the hardware also wants to write to the bit protected bit. Normally, the software will win and the 'old' value will be written to the bit protected bit. This is ok for bits which are not bit protected. For bit protected bits, we proceed differently. BPI checks for each bit whether its value has changed and provides a signal that indicates whether its value has changed. This is protect_bus_o[31:0]. A line of protect_bus_o going high indicates that the value of the corresponding bit has changed. A write access is now performed only for the bit where protect_bus_o indicates a new value. Thus, if the value of a bit is not changed by software (the case during a bit write to a different bit) then the hardware can write to this bit.

7. Interrupt Register/Node Handling

To maintain the full flexibility for the implementation of the interrupt node, the interrupt nodes are not included in the peripherals. This means that the interrupt registers are not part of the peripheral module. However, BPI delivers read and write signals for the interrupt registers that are related to the current peripheral. So we get as an output of the peripheral:
BPI_WR_<interruptregister>_N, BPI_RD_<interruptregister>_N.

8. Implementation and VHDL related stuff

Please find under the platform page on the intranet "VHDL Testbench Concept" by Thomas Hillman.

Please find under the platform page on the intranet "BPI Specification Draft" by Helmut Steinbach and Andreas Weisgerber.

Appendix P

Title:
BPI Specification
Draft Version 0.9

BPI Specification Draft Version 0.9

1. General

This document describes how to use the Bus-Peripheral-Interface. In the current version, this document is about BPI for FPI bus only.

2. Function

2.1. General concept

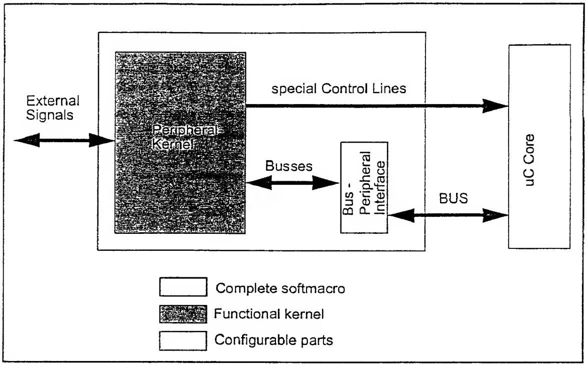
A Bus-Peripheral Interface BPI completely handles the bus protocol of the chip internal interconnect bus; this BPI can therefore be seen as the standardized interface to the FPI-Bus.

On the other side all BPIs adhere to the fundamentals given in "Platform Concept: SMIF Specification" (T. Steinecke, P. Schneider) and considered binding for all platform peripheral.

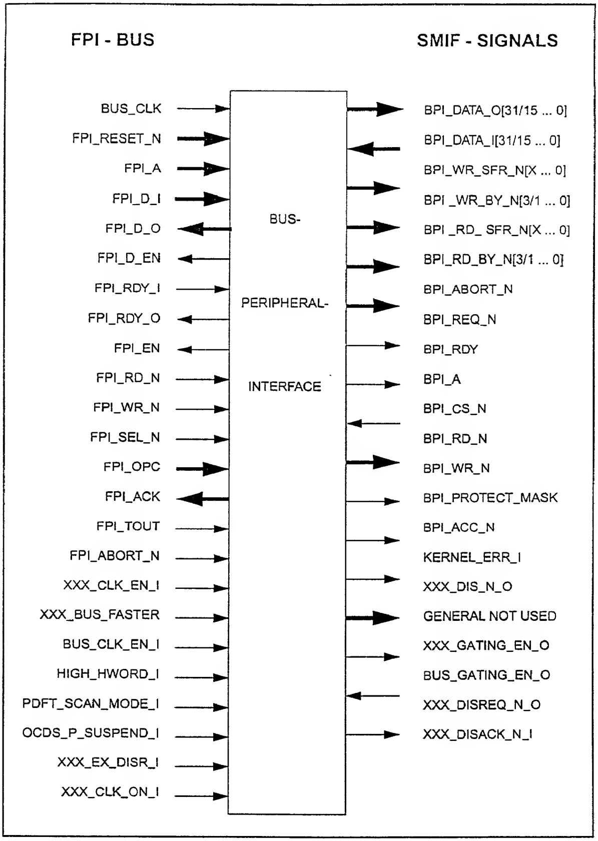
The peripheral's functional code (named "kernel" throughout this document) is thus independent of the used bus protocol.

The main tasks of the BPI are

- Buffering To define a predictable bus timing some standardisation on fan out and fan in is mandatory
- Decoding As Kernel SFR's will have different addresses in different uC's, addresses are decoded inside BPI mainly.
- Handshaking Bus and peripheral may use different clock rates (even dynamically !). Thus some way of synchronization has to be provided.
- Error Handling Access to undefined locations and unclocked peripherals are functions not regarded legal under normal circumstances; some error notification via the bus may be needed.
- Test Mode If special test modes are necessary the should be implemented likewise throughout a chip.


External Signals
Peripheral Kernel
special Control Lines
Busses
Bus - Peripheral Interface
BUS
uC Core
Complete softmacro
Functional kernel
Configurable parts

2.2. I/O signals from BPI_INTERFACE



The FPI signals are described in the in the FPI-Specification.

Description of the interface ports to the SMIF:

BUS_CLK	Corresponds to the fpi_clk
fpi_rdy_i	Corresponds to the fpi_rdy signal (fpi_rdy_b) from the FPI
fpi_rdy_o	Ready acknowlege from the BPI. Driven to the signal fpi_rdy_b if enabled with the fpi_en_o signal.
fpi_en_o	Enable signal for the fpi_rdy_o and fpi_ack_o tristate driver
fpi_ack_o	Acknowlege signal from the BPI. Driven to the signal fpi_ack(_b) if enabled with the fpi_en_o signal. **Note: Must be INOUT for synopsys test compiler**
fpi_d_i	Corresponds to the fpi_d from the FPI
fpi_d_o	Driven by the BPI internal data register. Driven to fpi_d(_b) if enabled with the fpi_d_en_o signal.
fpi_d_en_o	Enable signal for the fpi_d_o tristate driver
BPI_DATA_I[31/15...0]	Input data bus, either 32 or 16 bits wide. Source: Peripheral kernel port kernel_data_o
BPI_DATA_O[31/15...0]	Output data bus, either 32 or 16 bits wide. Destination: Peripheral kernel port kernel_data_i
BPI _RD_SFR(x...0)	Dedicated read signals, one each for each SFR.
BPI_RD_BY_N[3...0] (optional)	Selection which bytes have to be read
BPI _WR_ SFR(x...0)	Dedicated write signals, one each for each SFR.
BPI_WR_BY_N[3...0]	Selection which bytes have to be written
BPI_PROTECT_MASK [31/15...0] (optional)	Mask for bit-protection. ‚0' means that the BIT must not be changed ‚1' means that the BIT must be changed in the peripheral.
BPI_ABORT_N (optional)	This signal is directly mapped to the FPI-BUS-signal FPI_ABORT_N. Each peripheral can abort any access in this way. **In the current version this signal is not used by the interface!! This is wrong and will be resolved in the next version**
BPI_REQ_N (optional)	Indicates a valid access if dynamic waitstate insertion is required (handshake_c=1). This signal is reset by BPI_RDY.

BPI_RDY (optional)	Ready indication from the kernel after a dynamic waitstate insertion. While this signal is driven ,0‘, it is not possible to request a new access with the signal BPI_REQ_N. Exactly one waitstate is inserted if BPI_RDY is directly bridged to BPI_REQ_N in the kernel
BPI_A (optional)	Synchronized address bits (latched FPI_A) for the RAM-Interface. Only valid if a RAM-Interface is configured
BPI_CS_N (optional)	Chip select signal for the RAM-Interface
BPI_RD_N (optional)	Read signal for the RAM-Interface
BPI_WR_N (optional)	Write signal for the RAM-Interface
BPI_ACC_N	This signal indicates to the kernel when a Write or Read access will take place. It is active for the period of one “Master clock” before and during the rising edge of the peripheral clock. With this signal, multiple reads or writes to a peripheral register can be prevented. A normal read cycle doesn‘t need this signal, but if a register performes destructive read accesses this signal must be considered.
XXX_BUS_FASTER	Notifies the BPI module of a peripheral that the bus clock rate is currently higher than the peripheral clock rate. This speed indication is needed to perform zero Waitstate write accesses to the peripheral's internal registers whenever the Bus clock rate is equal to or slower than the peripheral clock rate. Connecting XXX_BUS_FASTER to static ‘1’ will insert exactly one waitstate into each access.
KERNEL_ERR	Error signal from the Peripheral Core to the BPI module signalling an error condition during a not allowed RD/WR access from BPI module to the peripheral core. Must be connected to ,0‘ if not used. **In the current version this signal needs to be driven before an access will be performed. This is not possible if the peripheral clock is turned off !**
OCDS_P_SUSPEND	Notifies the peripheral to stop the peripheral clock for debugging purposes.
xxx_ex_disr	Requests the peripheral to stop the peripheral clock.
Bus_clk_en	Enable signal for the BPI component clock domain. Is used for clock gating.
xxx_clk_en	Enable signal for the peripheral kernel clock domain. Is used for clock gating.
Bus_gating_en	Enable signal for clock gating of the peripheral clock domain.

xxx_gating_en	Enable signal for clock gating of the peripheral clock domain.
xxx_clk_on	Special function pin for RTC. Must be connected to ,1' if not used!
xxx_dis_n_o	Special function pin for RTC. Leave open if not used.
xxx_disreq_n	Disable request signal to the kernel to switch off the peripheral clock.
xxx_disack_n	Disable acknowledge signal from the peripheral kernel to allow switching off of the peripheral clock. Bridge this signal to xxx_disreq_n if no specific function is implemented in the kernel
high_hword	**This signal is only considered during write accesses if the fpi data bus width is 32 bits and the kernel data bus width is 16 bits.** High_hword = „0“ means that the lower 16 bits of the fpi bus carry valid data and are to be mapped to the kernel data bus. High_hword = „1“ means that the upper 16 bits of the fpi bus carry valid data and are to be mapped to the kernel data bus.

2.3. Architecture of the whole peripheral and testbench

A peripheral is a hierarchical construction (see Figure 2 below).
The peripheral kernel can be given either as a structural or as a rtl description.
The components <project>_bpi and <project>_kernel are contained within the module <project>_syn. (syn means synthesis unit)

Example: p3_wdt_bpi and p3_wdt_kernel comprise p3_wdt_syn.

On this project level, no tristate drivers are included, they are instantiated in module <project>_bus_driver.

Example: p3_wdt_bus_driver.

For the distribution of the master clock to the BPI and its kernel a special clock driver each will be instantiatied inside module <project>_clock_gating.

Example: p3_wdt_clock_gating.

The three modules <project>_bus_driver, <project>_clock_gating and <project> _syn are contained inside <project>

The dedicated testbench (<project>_tb) for this toplevel module consist of the peripheral itself (Example: p3_wdt)and the test units FPI (bus model) and <project>_IOC_top (input output control).

Example: p3_wdt_ioc_top.

The figure below shows the internal hierarchy of the bpi_interface and the other components.

Remark: All VHDL units should have a unique prefix in their name to signal their belonging to a certain peripheral. *p3_xxx* is used throughout this document as generic peripheral name.

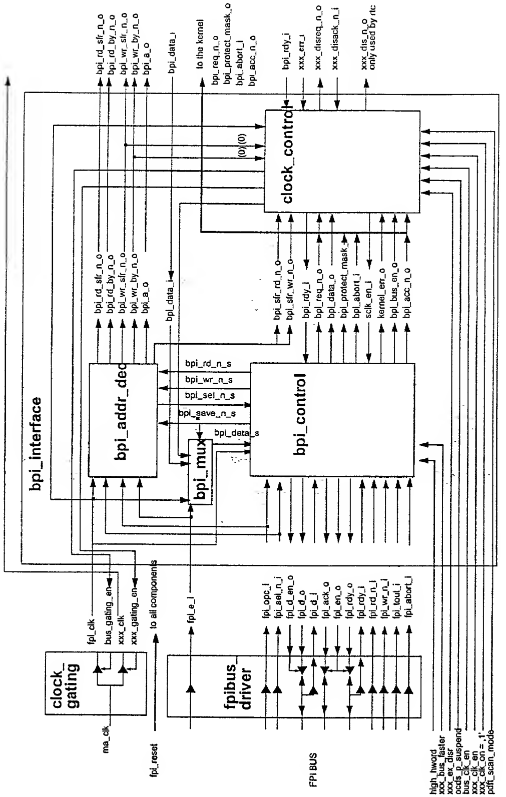


Figure 1: BPI architecture

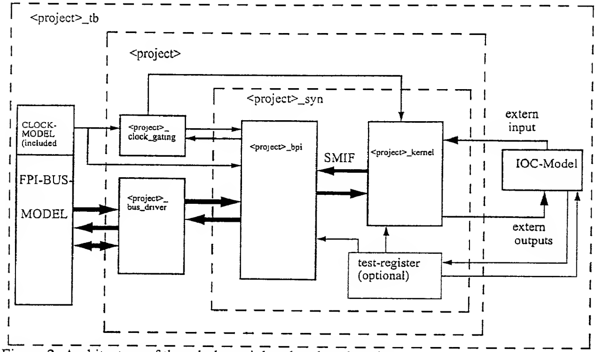


Figure 2: Architecture of the whole peripheral and testbench

If you use this example proceed as follows:

- Complete the component <project> and<project>_syn with the external input and output signals given by your<project>_kernel.
- Extend the signal list and modify the port map in file <project>_tbe_tba-a.vhd.
- Change the constant nr_of_pins_c and edit the constant pin map of the <project>_IOC_top port map

The <project>_tbe_tba-a.vhd contains an example.

3. Parameter setting in the project package

The configuration of the BPI is handled in the BPI_PACKAGE, contained in the file <project>_bpi_pack-p.vhd.
For every peripheral this package needs to be copied into the project packages directory under a unique name, usually by prefixing the original name with the project prefix, e.g. *P3_XXX*_BPI_PACK-P.VHD. Configuration of the BPI is then accomplished by editing (extending) this file. One important point is editing the name of the package likewise !

> Example for the watchdogtimer (WDT):
> $HWPROJECT/VHDL_PACKS/RTL/P3_WDT_BPI_PACK-P.VHD.

The personalised project package is then imported by all other units with the statement

USE WORK.*P3_XXX*_BPI_PACK.ALL;

Inside this package several constants are to be checked and changed if necessary:

fpi_addressbuswidth_c	Indicates the width of the incoming FPI_A - BUS Preset to 32 Bit. Normally no change necessary
fpi_databuswidth_c	Indicates the width of the bidirectional FPI_D - BUS Preset to 32 Bit. Normally no change necessary
kernel_databuswidth_c	Indicates how many bits are provided by the largest peripheral register in one access. Possible values are 32, 16, 8. The constant sfr_size_c indicates the numbers of bytes and is calculated from kernel_databuswidth_c by: sfr_size_c := kernel_databuswidth_c / 8
addressbuswidth_c	The constant addressbuswidth_c indicates how many bits are to be decoded by the address decoder for the SFR select signals. In order to get the correct address of the register the bits fpi_a(fpi_high_c downto fpi_low_c) will be decoded. Note that fpi_a(1 downto 0) are reserved for the byte selection. The default value is 8; {(7 downto 0)}
fpi_low_c	LSB of the decoded address for the SFR addressrange. Preset to 2. Normally no change necessary
fpi_high_c	MSB of the decoded address for the SFR addressrange. Preset to 7. Normally no change necessary
handshake_c	When you need a handshake-procedure between interface and kernel, set the constant handshake_c:=1; Preset to 0
fpi_addr_width_c	Range for bit_vector. Preset to 8. No change necessary unless the constant addressbuswidth_c was changed. This constant is only used to define the type of the following constant sfr_addr_c.
sfr_addr_c	Array of SFR-Addresses for the address decoder. Note that the first 4 addresses (x'00,x'04,x'08,x'0C) are predefined !
destructive_read_c	Indicates whether the peripheral includes destructive read registers. Preset to 0

dest_addr_c	Subarray of SFR-Addresses (must also be present in sfr_addr_c) which are destructive read registers. Preset to „10“. This constant is only relevant if destructive_read_c = 1. **If only one register is dest.read insert it twice**

The next group of constants is relevant if a RAM-interface is required only.

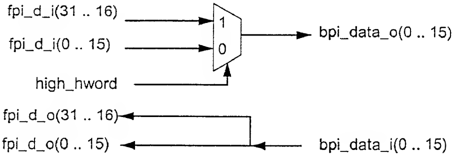
ram_interface_c	Gobal indication if a RAM is required! Preset to FALSE
ram_addr_c	The constant indicates the bit combination which shall create the select signal for the ram inside the kernel. Preset to „1“ This constant is compared to fpi_a(ram_high_c downto ram_low_c) **If ram_high_c is not equal to ram_low_c define a subtype !**
ram_addr_low_c	Indicates the lsb of the RAM address. Preset to 0
ram_addr_high_c	Indicates the msb of the RAM address. Preset to 7 bpi_a<=fpi_a(ram_addr_high_c downto ram_addr_low_c) Preset to bpi_a<=fpi_a(7 downto 0)
ram_low_c	LSB of the decoded ram address. Preset to 8. Normally no change necessary
ram_high_c	Preset to 8 ; -- only 256 byte RAM If you set this constant, for example to the value 9 then don't forget to change the constant ram_addr_c to a 2-Bit-Vector. Example: std_ulogic_vector(ram_high_c downto ram_low_c) := „10“ This sector indicates if a RAM or SFR are selected.
ram_databuswidth_c	Preset to 32 -- the same as the FPI data bus. You can change it to 16 or 8 for another ram databuswidth. **!! IN THE CURRENT VERSION !!!!** **!! ram_databuswidth_c must equal kernel_databuswidth_c!** **!! It is no own data input for the RAM inserted !!**
ram_access_c	The result of the calculation ram_databuswidth_c / 8

If a RAM-module is not configured (ram_interface_c = FALSE), the RAM special signals must not be used in the kernel. These are .

BPI_A_I	the registered fpi_a(ram_low_c-1 downto 0)
BPI_WR_N_I	write signal for the RAM
BPI_RD_N_I	read signal for the RAM
BPI_CS_N_I	chipselect signal for the RAM

4. Binary and Enhanced Mode

For compatibility reasons some bus mapping is provided. This is controlled by the special signal high_hword. Only if the FPI databus width is 32 and the Kernel databuswidth is 16 is it possible to multiplex the upper or the lower16 Bit of the FPI databus to the kernel databus independently from the lsbs of fpi_a ! Nomally this signal must be held '0'.



5. The FSM of the controlunit

5.1. control_m1

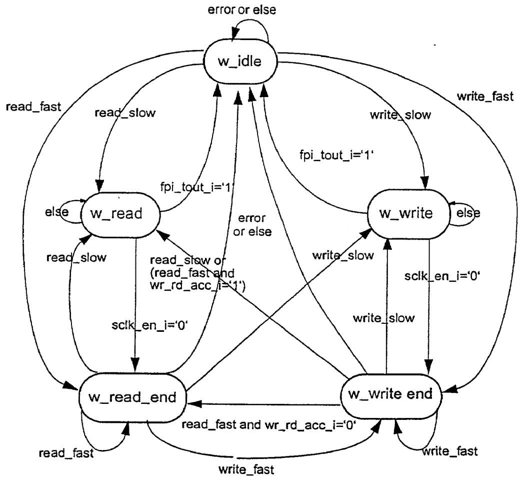
read_fast <= xxx_bus_faster_i='0' and fpi_rd_n_i='0' and fpi_opc_i /= „1111" and
fpi_sel_n_i='0' and fpi_rdy_i='1' bpi_sel_n_i='0';

write_fast <= xxx_bus_faster_i='0' and fpi_rd_n_i='1' and fpi_opc_i /= „1111" and
fpi_sel_n_i='0' and fpi_rdy_i='1' bpi_sel_n_i='0' and fpi_wr_n_i='0';

read_slow <= (xxx_bus_faster_i='0' and fpi_rd_n_i='0' and fpi_opc_i /= „1111" and
fpi_sel_n_i='0' and fpi_rdy_i='1' bpi_sel_n_i='0')
or (read_fast and bpi_dest_i='1';

write_slow <= xxx_bus_faster_i='0' and fpi_rd_n_i='1' and fpi_opc_i /= „1111" and
fpi_sel_n_i='0' and fpi_rdy_i='1' bpi_sel_n_i='0' and fpi_wr_n_i='0';

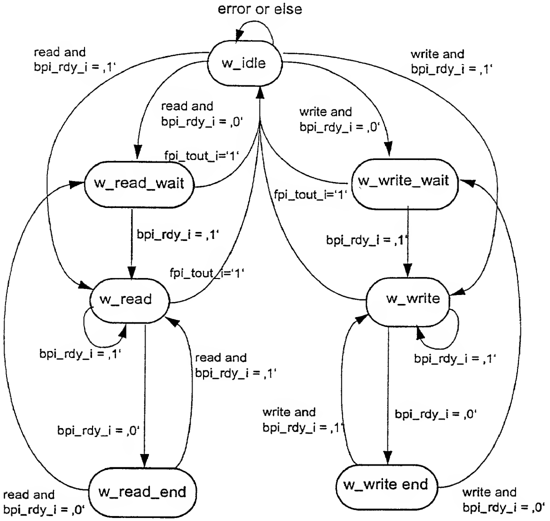
error <= fpi_tout_i='1' or kernel_err_i='1';



5.2. control_h (handshake)

read <= fpi_rd_n_i='0' and fpi_opc_i /= „1111" and
fpi_sel_n_i='0' and fpi_rdy_i='1' bpi_sel_n_i='0'

write <= fpi_rd_n_i='1' and fpi_wr_n_i='0' and fpi_opc_i /= „1111" and
fpi_sel_n_i='0' and fpi_rdy_i='1' bpi_sel_n_i='0'



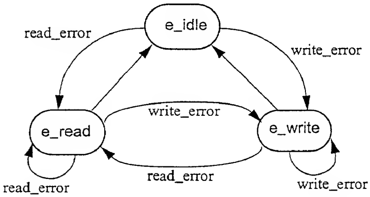
5.3. FSM for error handling

An error occured if

- the peripheral is selected but the SFR address is wrong,
- a access is performed while the kernel clock is switched off or
- a access is performed while the kernel indicates a kernel error

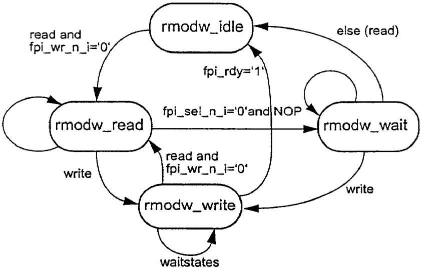
In the e_read state the fpi data bus nonetheless must driven by the peripheral.
If the kernel clock is switched off only accesses to the clock control register is allowed.

In case of timing problems there is a constant error_handling_c **defined inside the architecture p3_xxx_bpi-rtl-a.vhd to switch off this errorhandling** (constant error_handling_c: boolean:=false;)**! In this case the fpi_ack is in everytime NSC!**

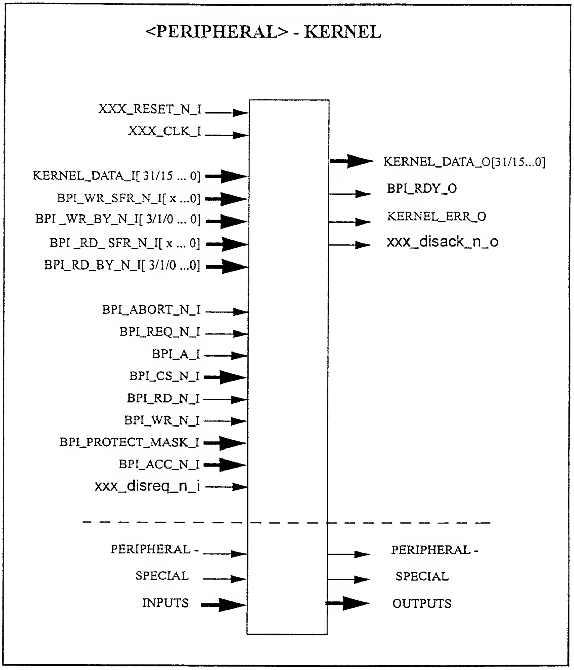


5.4. The read modify write FSM

The rmodw FMS rememberes the read part of a modify access and waites for the corresponding write access.



6. I/O signals from Peripheral Kernel



The signals have already been described above.
If no error indication from the kernel is required, two signals must be driven nonetheless by the architecture of the kernel:

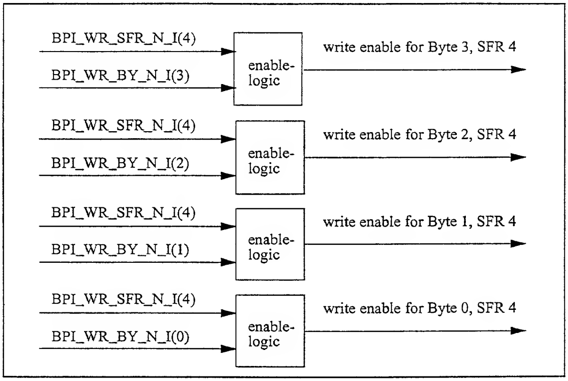
kernel_err_o <= ,0‘; -- special error from kernel
xxx_disack_n_o <= xxx_disreq_n_i; -- always ok to disable clock

7. How you can use the port signals

7.1. Write enable logic

A register normally consist of one, two or four bytes. In order to write each byte separately the BPI provides a dedicated enable signal (BPI_WR_SFR_N_I(x)) for each SFR and a separate enable signal (BPI_WR_BY_N_I(3..0)) for each byte of the data bus. For each byte of each register the write enable will be generated from this signals.
If the signal BPI_WR_SFR_N_I of the SFR and all signals of the BPI_WR_BY_N_I are low, then the whole register is enabled to store the new data.

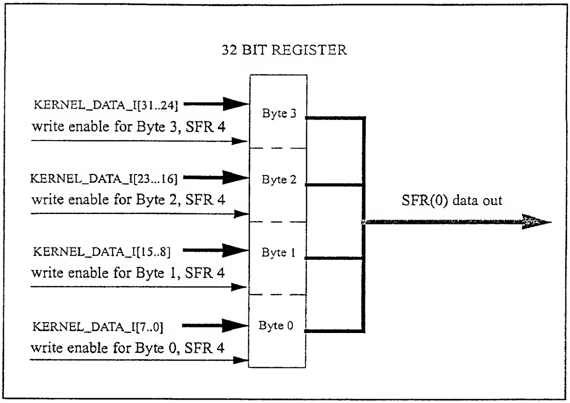
The figure below illustrates the proposed circuit



Caution!!
The BPI_WR_SFR_N_I[0 .. 3] and BPI_RD_SFR_N_I[0 .. 3] are reserved normally !!

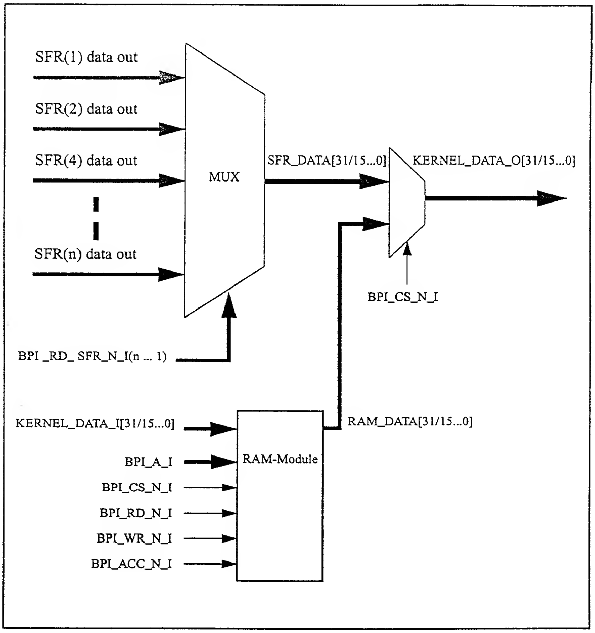
SFR(0) is the clock control register and is instantiated in the BPI.
SFR(1) is the peripheral input select register
SFR(2) is the identification number
SFR(3) is reserved for future use

The correlation between index and address is given by the order of the addresses in the addresslist in the bpi_package.
Note that only SFR(0) is implemented inside the BPI's architectures (clc-entity)


32 BIT REGISTER
KERNEL_DATA_I[31..24]
write enable for Byte 3, SFR 4
Byte 3
KERNEL_DATA_I[23...16]
write enable for Byte 2, SFR 4
Byte 2
SFR(0) data out
KERNEL_DATA_I[15..8]
write enable for Byte 1, SFR 4
Byte 1
KERNEL_DATA_I[7..0]
write enable for Byte 0, SFR 4
Byte 0

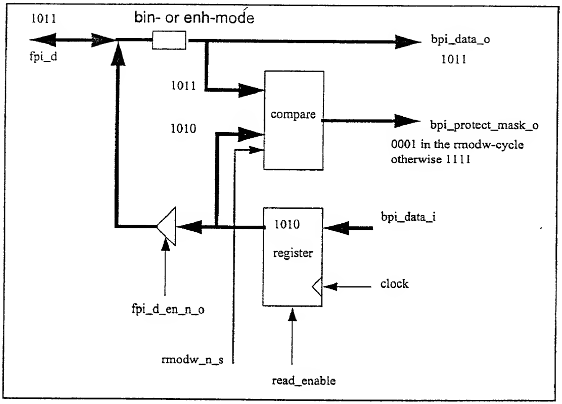
7.2. Multiplex output data

The next figure shows one possibility to propagate the output data of the kernel. Note that multiplexing RAM data and SFR data in two stages is only an example.

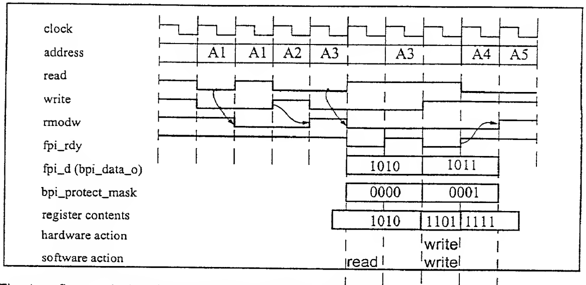


7.3. Read modify write cycle and protect mask

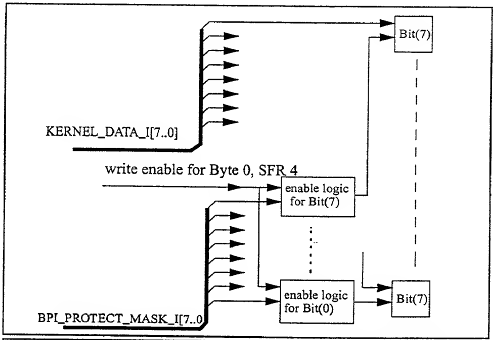
Read-Modify-Write is coded on the FPI bus and recognized by the BPI interface. The read modify write signal is not forwarded to the kernel.
As a result a signal RMODW_N_S is low active for the complete read modify write cycle on the FPI-BUS. As no other SFR may be read intermittingly the read data can be stored in the read register. A protection mask can now be derived by comparing the (locally stored) read data with the new data to be written. All bits not changed are given a mask value 0 if the RMODW_N_S signal is active.
In a normal write or read cycle, all bits of the mask are „1“.
Protect mask bit equal to „1“ means that the data bit must be written into the register.
Protect mask bit equal to „0“ means that the data bit must not be written into the register.
This is illustrated by the next figures.

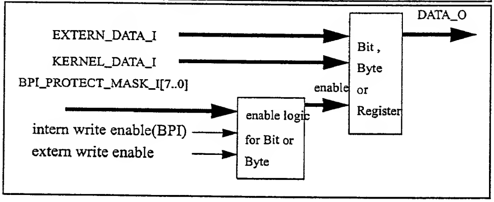


The bpi_data_i are only captured at the end of the read-cycle and remain unchanged by the write- and idle-cycles on the FPI bus. The protect mask is provided every time. Only register bits which can be modified by SW and HW need to use the protection mask.



The two figures below illustrate how to use these signals



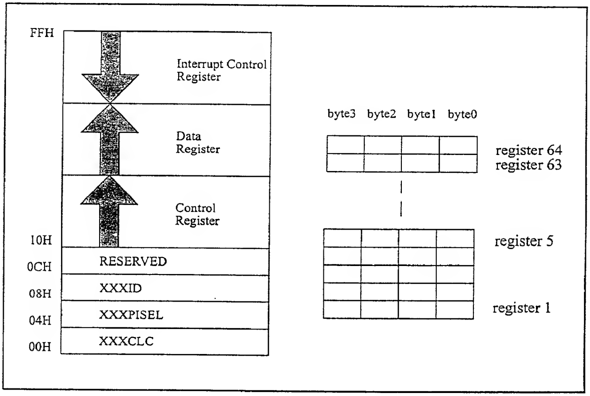


8. Peripheral Address Map

Each peripheral has n x 256 Byte register blocks or n x 64 32-Bit register. The register is selected with signals BPI_WR_SFR_N_I or BPI_RD_SFR_N_I . Four registers are fixed:

- Address 00H is the peripheral clock control register.
- Address 04H is the peripheral port input select register.
- Address 08H is the peripheral identification. No real register, hard coded.
- Address 0CH is reserved.

Control registers start at address 10H, followed by the data register.
The interrupt control register start at the address FFH decreasing order.



9. Usage of the fixed Registers

9.1. XXXID - REGISTER

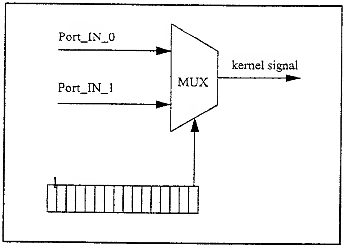
The XXXID is not a real register, it is hard coded and readable only.
Example :

```
        perid_s <= „10100111“;
                  or
        perid_c : std_ulogic_vector(8 downto 0) := „10100111“;
```

It is selected with bpi_rd_sfr_n_i(2).
It should be assured by appropriate dont_touch attributes that the ID is implemented on silicon as matel contacts. That allows cange of revision for metal redesigns.

9.2. XXXPISEL - REGISTER

The XXXPISEL register indicates which port shall be used for kernel inputs.
The register has to be implemented by the peripheral designer and is selected with bpi_wr_sfr_n_i(1). Every bit in this register controls one multiplexer.
Therefore, every external input of the kernel can be driven by one of two sources.
Example :
If XXXPISEL(0) = ‚0' then kernel signal <= Port_0 else kernel signal <= Port_1;



9.3. XXXCLC - REGISTER

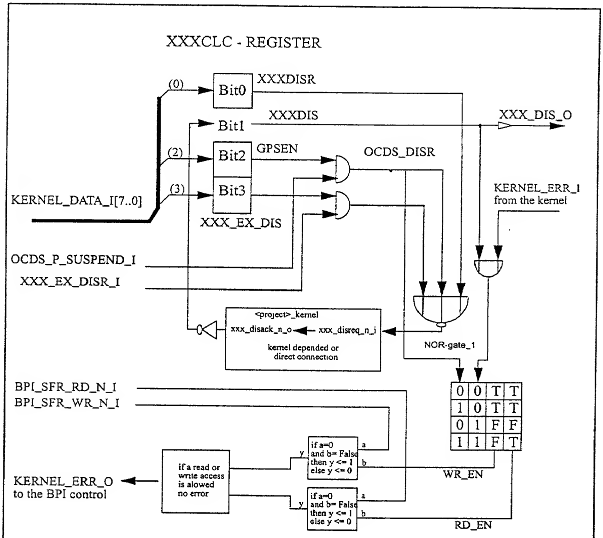
The state of the peripheral clock is controlled by a register bit "XXXDISR" located in the register XXXCLC of the peripheral core. This register XXXCLC is clocked with the bus clock to be able to switch the peripheral clock on again if it was off. If required by the peripheral's functionality switching off the clock can be prevented by the peripheral. The actual clock state will be shown by the state bit "XXXDIS" within the same register. The signal "XXX_DIS_N" which is a combinatorial combination of other clock control signals represents the actual enable state of the peripheral clock and is used to switch on/off the peripheral clock. The clock gating buffer is located in the Clock Gating block of a peripheral. The clock enable signals used to control the clock speeds are generated in the central Clock Generation and distributed to the Clock Gating block.
The BPI module rejects every FPI bus access with Error condition if the peripheral clock is switched off (signaled by the signal Kernel_err_o- driven by clc module, not peripheral kernel).

9.3.1. Coherence between XXXCLC and the OCDS_P_SUSPEND, XXX_EX_DISR

For On Chip Debugging support an additional signal OCDS_P_SUSPEND is intro-

duced to stop the peripheral clock for debugging if this function is enabled. If debugging mode is active the peripheral core rejects write access to registers connected to the peripheral clock by activating the signal Kernel_err. This causes the BPI module to reject FPI bus accesses with Error condition. Read accesses to registers of the peripheral clock domain are possible.
To be compatible with old products an XXX_EX_DISR signal is introduced to disable the peripheral clock.

For more information see also the **Peripheral Clock Strategy Specification**



clc data output is Bit(3 downto 0)

10. Forwarding

If a read access immediately follows a write cycle, it is normally not possible to read the previously written data in a zero waitstate access due to the required synchronisation. To resolve this problem no forwarding multiplexer is provided; instead a waitstate will be inserted if a read access is performed after a write access to the same register and without any idlestates in between !

11. Timing Diagrams

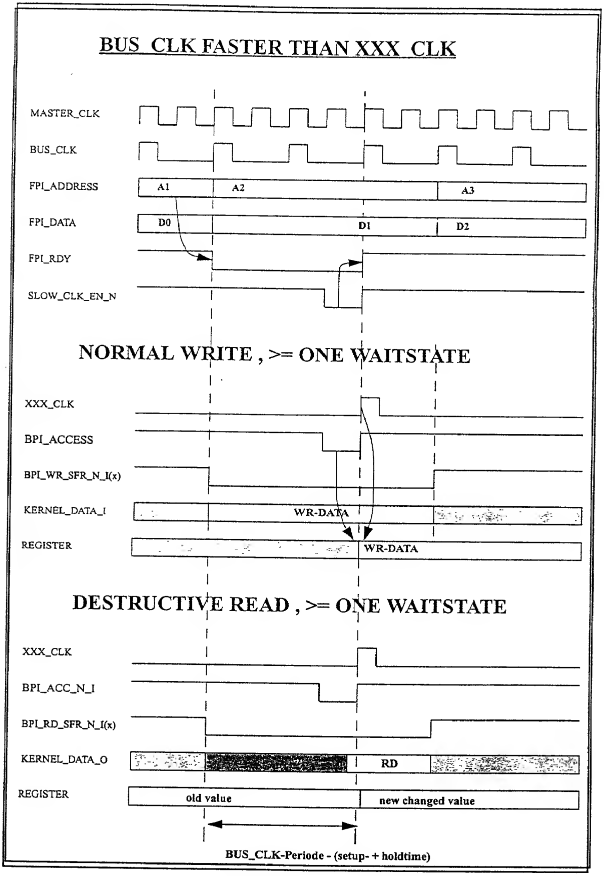
As there is no fixed relationship between bus clock and peripheral clock several different timing diagrams are provided.

The first possibility is, that the bus clock is faster (bus_faster is ,1') as the peripheral clock.

The second possibility is, that the bus clock is slower tahn or exactly as fast as the peripheral clock (bus_faster is ,0').

If it's necessary to implement a RAM inside the kernel the cycle must start with one waitstate. This is not implemented!

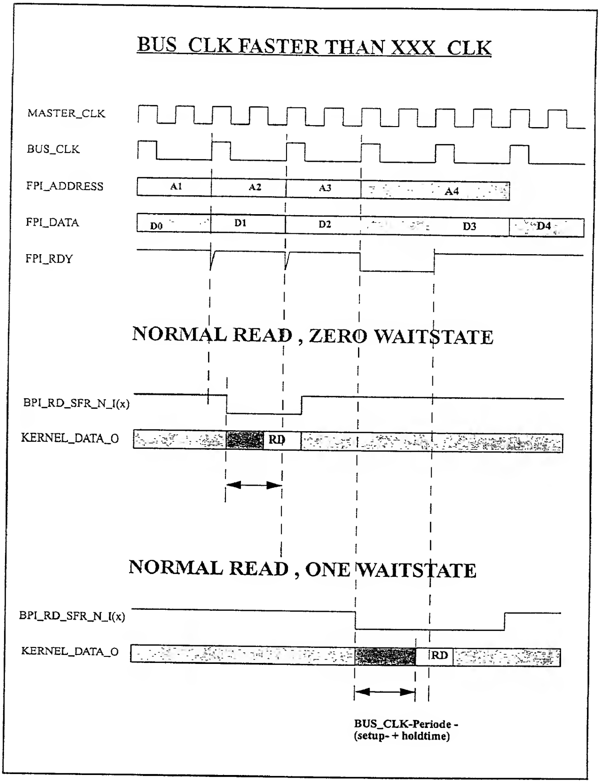
The BPI_ACC_N_I signal is used to avoid multiply writes to the same address due to to fast a peripheral clock. It will be driven low activ one clockcycle before the relevant rising edge of the peripheral clock. It is activated whenever a register must store the data in the write cycle or change the data after a destructive read cycle.



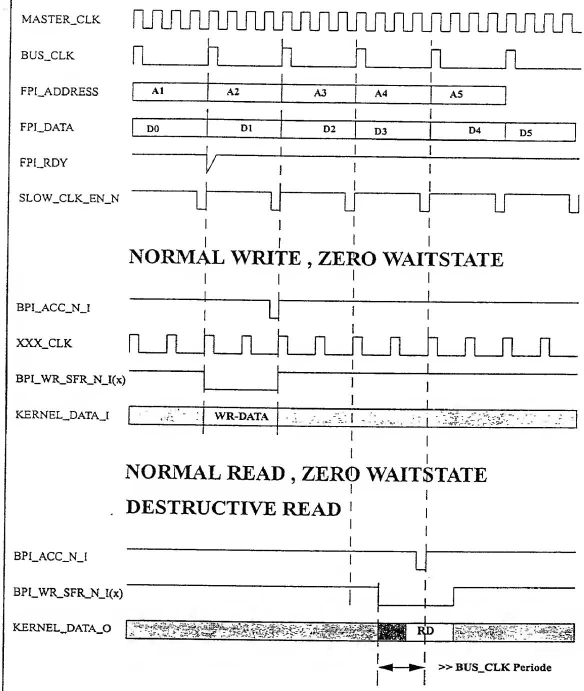
BUS CLK FASTER THAN XXX CLK

NORMAL WRITE , >= ONE WAITSTATE

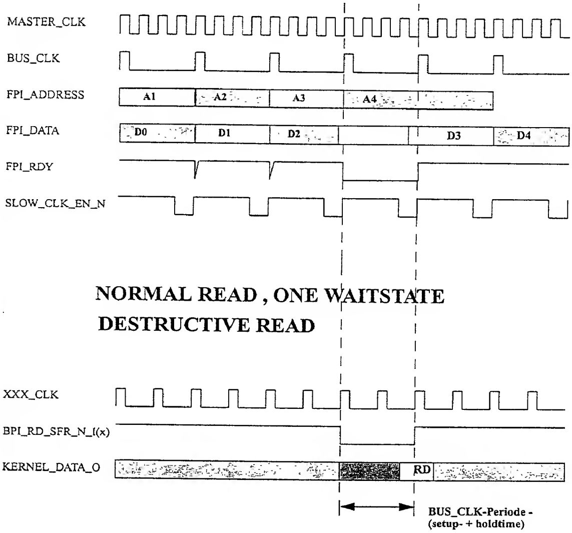
DESTRUCTIVE READ , >= ONE WAITSTATE


BUS CLK FASTER THAN XXX CLK
MASTER_CLK
BUS_CLK
FPI_ADDRESS
A1
A2
A3
A4
FPI_DATA
D0
D1
D2
D3
D4
FPI_RDY
NORMAL READ , ZERO WAITSTATE
BPI_RD_SFR_N_I(x)
KERNEL_DATA_O
RD
NORMAL READ , ONE WAITSTATE
BPI_RD_SFR_N_I(x)
KERNEL_DATA_O
RD
BUS_CLK-Periode -
(setup- + holdtime)

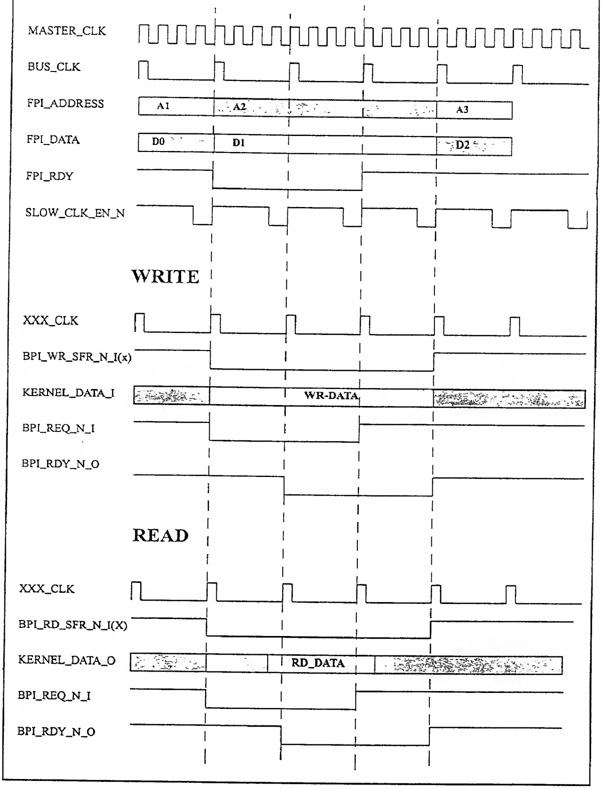
BUS CLK SLOWER OR EQUAL THAN XXX CLK



BUS CLK SLOWER OR EQUAL THAN XXX CLK

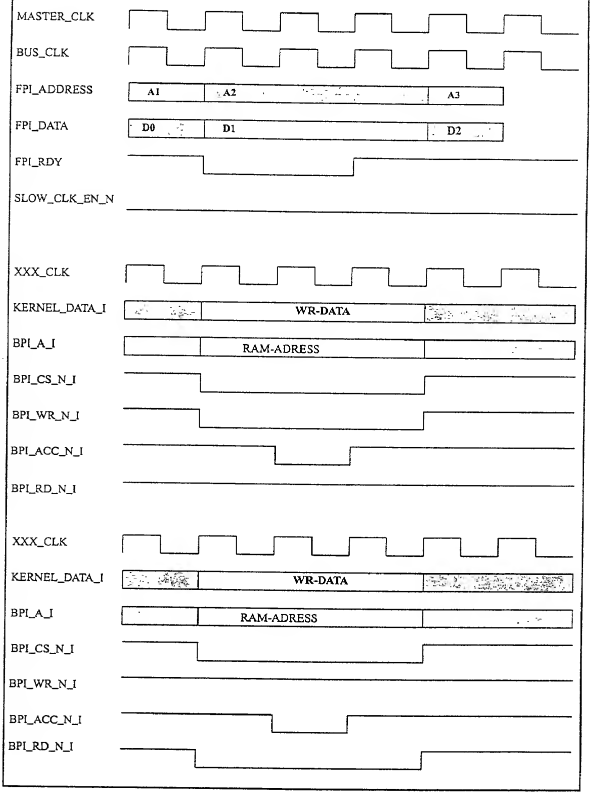


HANDSHAKE BETWEEN INTERFACE AND KERNEL

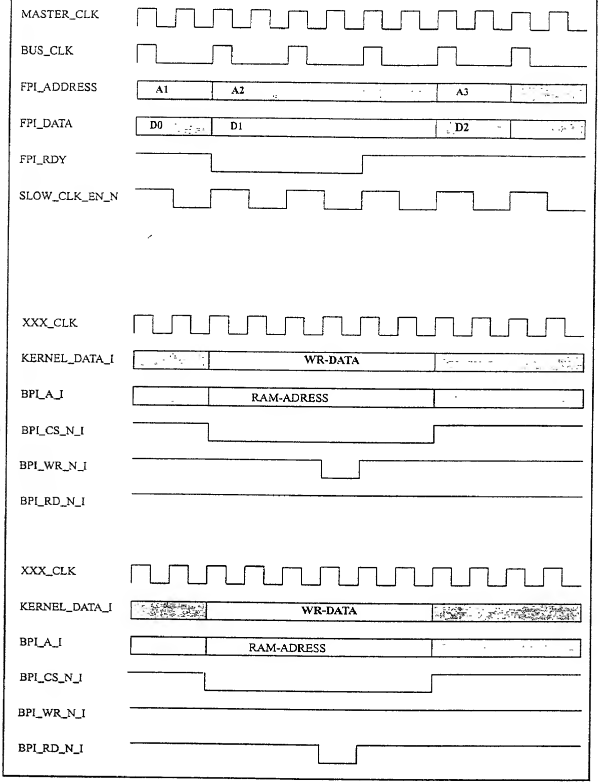


RAM INTERFACE (not implemented)

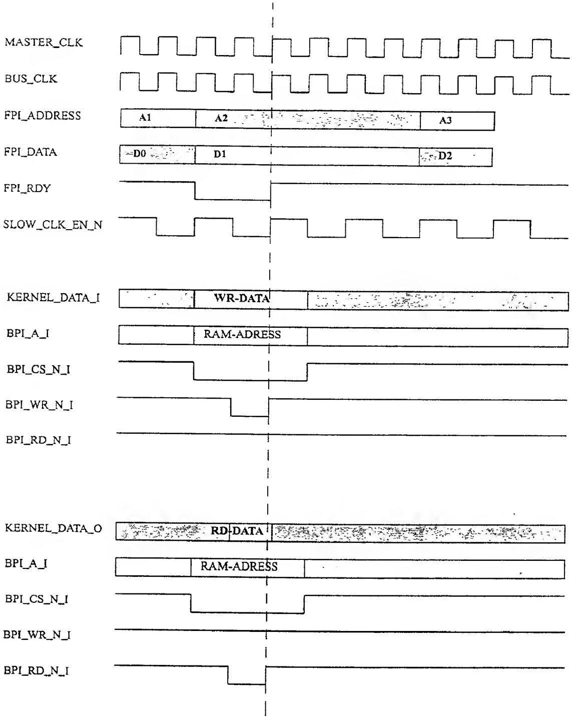
The current version is like a nomal access and provides the RAM signal but not in the timing below



RAM INTERFACE - BUS SLOWER THAN KERNEL



RAM INTERFACE - BUS SLOWER THAN KERNEL



12. Module Developers Test Cook Book

This chapter is a copy of Hermann Obermeir's paper regarding this topic; contact him for updates !

Platform peripherals are prepared for ATPG and for isolated module test. The following gives a brief overview what a module developer has to do.

12.1. Automatic Test Pattern Generation(ATPG)

12.1.1. Prepare setup files

e.g. GNUmakefile.setup
INSERT_SCAN_DESIGN := true
SCAN_CHAINS := <desired_number>

12.1.2. Run check test

- In Design Analyzer: Tools -> Test Synthesis: Check Design Analyzer
- you should have no violations

12.1.3. Insert Scan Registers and generate pattern

Use "ssemake insert_scan" to insert scan registers.
Build scan chains up to a maximum length of 100 flipflops.
serial inputs of the scanchains: **pdft_sci_0_i, pdft_sci_1_i,.....**
serial outputs: **pdft_sco_0_o, pdft_sco_1_o,....**
scan enable input: **pdft_scen_i**

12.1.4. Generate test Pattern

"ssemake atpg" You should achieve 100% fault coverage.
(If not make sure with a fault simulation of your module test pattern, that the untested faults are tested with your module test pattern)

12.1.5. Save Results

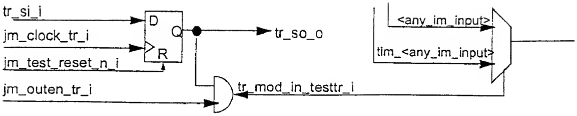
Save Results (Fault coverage,unvovered fauts)
Save constraints for the ATPG tool(set_test_hold, set_test_assume), if they have been used.

12.2. Preparation for isolated module test

The platform peripherals are tested in an isolated module test. The peripheral signals are made transparent at the chip boundary e.g. using multiplexers.
There are three categories of pins:
FPI bus: the FPI bus is made transparent via EBC/EBU
Alternate I/Os are made visible via normal prot functions
Outputs to other modules (here called intermoduleoutputs) are multiplexed in the ports
Inputs from other modules are multiplexed in the peripherals (intermoduleinputs)

12.2.1. Insert Testregister and Multiplex Inputs

Usually 1 test register bit is required. Use .... as an example
All InterModuleInputs are fed into a multiplexer and a testinput tim_.... is created, which will be connected to external pin on chip level



any_im_input = any_intermodule_input
tim = prefix for "test isolatd module"

tr_si_i, jm_clock_tr_si, jm_test_reset_n_i, jm_outen_tr_si are module inputs
tr_so_o is a module output. They will be connected at chip level

Further Cases:

- No test register: Some modules (SSC, ASC, IIC) have no dedicated intermodule inputs. In this cases the test register may be omitted. Such execptions have to be approved bye Georg Sigl, because they reduce testability.
- Addtional test register bits are necessary, if the module increases IDDQ-current (e.g. pullups, pulldowns) or the test register needs to be in a special mode for testing other peripherals (e.g. for clock generation)

12.2.2. Insert MISR

Insert Signature Registe Kann es je vorkommen, dass man das Signaturregister im Betrieb mitlaufen lassen moechte, dann braeuchten wir ein eigenes testregister Bit, ansosnter gemeinsam nit Eingansgsmultiplex

12.2.3. Insert MISR

You need a MISR, if your module has many inter module outputs (more than 5 to 10)

- Generate MISR using genbist (length is number of your intermodule outputs, but minimum 20
- add an asynchronous reset to the misr: jm_test_reset_i
- the bcode inputs of the MISR are test pins to your module: pdft_misr_bcode[1:0]
- the intermoduleoutputs are fed into the parallel MISR inputs
- the msb ofthe misr will be a test
- in test mode of the peripheral the MISR follows is controlled by the bcode test pins, in functional mode a synchronous reset is applied to bcode (bcode[1:0] ='10'
-

12.2.4. Select Module Pattern

The module pattern are used for a functional/performance test of the modul.
select a set of PDL files and store them.
Add a comment into your PDL file, where
If ATPG could not achieve 100% fault coverage make these

12.3. IDDQ Test preparation

IDDQ test vectors will be selected for platform products using Viewlogic/Sunrise. All peripheral have to fulfill the testability rules for this tool. Its not yet detemined, if we have to check every peripheral with Sunrise/testability checker.

12.4. Fault Coverage

12.5. Document Testability in Module Specification

Document Testability in your Module Specification. Use SSC as an example

13. Product Developers Test Cook Book

tbd

13.1. Preparations

13.1.1. Port Description

A description of the ports in computer readable format is necessary

13.2. Make Top Level Entity

13.2.1. Wiring of Test Register

13.2.2. Wiring of Scan Registers

13.3. Make Testbench

The necessay testbench contains

Infineon technologies

TD6D4D" 45952560

HARRIER: Security Building Blocks

EXHIBIT Q
Page 1

[redacted]

[redacted]

Dual State Transition Logic sArbiter sADRDEC	<u>Dual State Transition Logic</u> ❑ secure and non-secure states of operation ❑ Secure Address Decoder ❑ Secure Bus Arbiter
Security Cache Logic (3DES) 4k Buffer	<u>Secure Cache Logic</u> ❑ secure MMU functionality ❑ Supports externally encrypted code
IPSec (DES, 3DES, MDS, SHA-1)	<u>IP Sec Processor</u> ❑ Parallel DES/3DES & hash operations ❑ Supports 2-key & 3-key 3DES ❑ Supports SHA-1 and MD5 hash functions ❑ Supports HMAC functionality ❑ DMA interface
Exponentiator	<u>Exponentiator/Multiplier</u> ❑ Accelerates public key operations
Security Assurance Logic	<u>Assurance Logic (Electrical Attack Prevention)</u> ❑ Programmable voltage protection ❑ Programmable frequency-out-of-range detection ❑ Power-on-reset ❑ Single event upset detection ❑ Active Zeroization ❑ Watchdog timer ❑ Reset protection
Randomizer	<u>Random Number Generator</u> ❑ Fully digital and non-deterministic (FIPS 140-1 compliant)
RTSC	<u>RealTime Seconds Counter</u> ❑ 32-bit counter (120 years) ❑ Adjsutable time base
Firmware	<u>On-chip Firmware</u> ❑ IPSec and IKE support ❑ IPSec as BITS ❑ IPSec header insertion ❑ AH/ESP in tunnel/transport mode ❑ IKE ❑ Key management ❑ Security associations ❑ RTOS compatibility

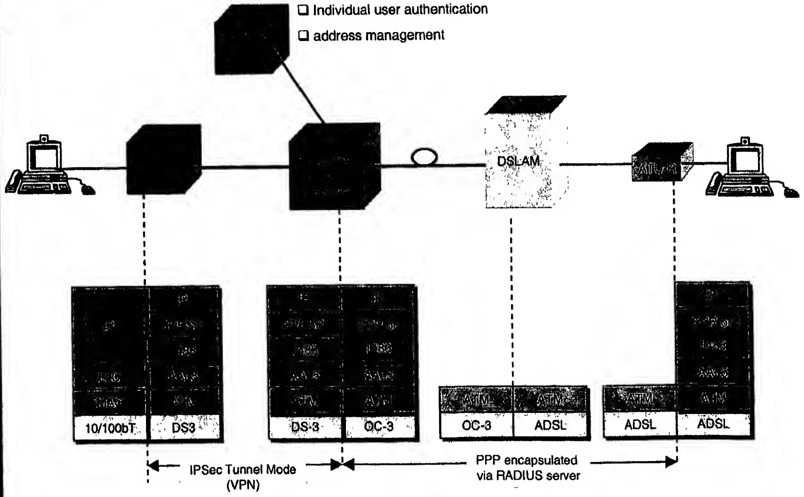
Exhibit Q

Page 2

xDSL: IPSec Tunneling

DRAFT VERSION

CONFIDENTIAL



□ Individual user authentication

□ address management